symmetry

MDPI

Article

Multimedia Data Modelling Using Multidimensional Recurrent Neural Networks

Zhen He [1,2,*], Shaobing Gao [3,*], Liang Xiao [4], Daxue Liu [1] and Hangen He [1]

1. College of Intelligence Science, National University of Defense Technology, Changsha 410073, China; daxueliu@126.com (D.L.); hehangen2000@163.com (H.H.)
2. Department of Computer Science, University College London, London WC1E 6BT, UK
3. Department of Computer Science, Sichuan University, Chengdu 610065, China
4. Unmanned Systems Research Center, National Innovation Institute of Defense Technology, Beijing 100071, China; xiaoliang.cs@gmail.com

* Correspondences: zhen.he@ucl.ac.uk (Z.H.); gaoshaobing@scu.edu.cn (S.G.); Tel.:+86-139-7514-0773 (Z.H.); +86-187-8220-1523 (S.G.)

Received: 19 August 2018; Accepted: 27 August 2018; Published: 1 September 2018

Abstract: Modelling the multimedia data such as text, images, or videos usually involves the analysis, prediction, or reconstruction of them. The recurrent neural network (RNN) is a powerful machine learning approach to modelling these data in a recursive way. As a variant, the long short-term memory (LSTM) extends the RNN with the ability to remember information for longer. Whilst one can increase the capacity of LSTM by widening or adding layers, additional parameters and runtime are usually required, which could make learning harder. We therefore propose a *Tensor LSTM* where the hidden states are tensorised as multidimensional arrays (tensors) and updated through a *cross-layer convolution*. As parameters are spatially shared within the tensor, we can efficiently widen the model without extra parameters by increasing the tensorised size; as deep computations of each time step are absorbed by temporal computations of the time series, we can implicitly deepen the model with little extra runtime by delaying the output. We show by experiments that our model is well-suited for various multimedia data modelling tasks, including text generation, text calculation, image classification, and video prediction.

Keywords: multimedia data modelling; recurrent neural network (RNN); long short-term memory (LSTM); tensor; convolution; deep learning

1. Introduction

Multimedia data such as text, images, and videos are ubiquitous nowadays. Modelling such data usually involves the analysis, prediction, or reconstruction of them. For instance, text modelling relates to many natural language processing tasks such as sentiment analysis [1], part-of-speech tagging [2], machine translation [3], and question answering [4], image modelling relates to many computer vision tasks such as image segmentation [5], depth reconstruction [6], image generation [7], and super-resolution [8], and video modelling also relates to many computer vision tasks such as object tracking [9], video segmentation [10], motion estimation [11], and video prediction [12]. Although they are diverse, these tasks usually can be formulated as a time series prediction problem, e.g., generating a desired output $\boldsymbol{y}_t$ for a given time series $\boldsymbol{x}_{1:t} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_t\}$, for time $t = 1, 2, \ldots, T$, where $\boldsymbol{x}_t \in \mathbb{R}^U$ and $\boldsymbol{y}_t \in \mathbb{R}^V$ are vectors (In this paper, we assume the vectors are in row form). The recurrent neural network (RNN) [13,14] is a popular model that can learn to encapsulate the useful information of the input history $\boldsymbol{x}_{1:t}$ into a hidden state vector $\boldsymbol{h}_t \in \mathbb{R}^M$. By concatenating the input $\boldsymbol{x}_t$ to the previous hidden state $\boldsymbol{h}_{t-1}$, we first get $\boldsymbol{h}^{cat}_{t-1} \in \mathbb{R}^{U+M}$:

$$\boldsymbol{h}^{cat}_{t-1} = [\boldsymbol{x}_t, \boldsymbol{h}_{t-1}]. \tag{1}$$

The hidden state $\boldsymbol{h}_t$ is then updated by:

$$\boldsymbol{a}_t = \boldsymbol{h}^{cat}_{t-1}\boldsymbol{W}^h + \boldsymbol{b}^h, \tag{2}$$
$$\boldsymbol{h}_t = \phi(\boldsymbol{a}_t), \tag{3}$$

where $\boldsymbol{W}^h \in \mathbb{R}^{(U+M)\times M}$ and $\boldsymbol{b}^h \in \mathbb{R}^M$ are parameters namely the *weight* and *bias*, respectively; $\boldsymbol{a}_t \in \mathbb{R}^M$ is the activation for $\boldsymbol{h}_t$, and $\phi(\cdot)$ is the tanh function (element-wise). The RNN finally produces an output $\boldsymbol{y}_t$ for time t:

$$\boldsymbol{y}_t = \varphi(\boldsymbol{h}_t\boldsymbol{W}^y + \boldsymbol{b}^y), \tag{4}$$

where $\boldsymbol{W}^y \in \mathbb{R}^{M\times V}$ and $\boldsymbol{b}^y \in \mathbb{R}^V$, and $\varphi(\cdot)$ is a differentiable transformation that depends on the task.

Nevertheless, the standard RNN is notorious for capturing the long-term dependency caused by the vanishing and exploding gradients [15]. Long Short-Term Memories (LSTMs) [16,17] mitigate this by (i) introducing the memory cell to store information longer, and (ii) utilising the gates for information routing. In a standard LSTM [17], the hidden state $\boldsymbol{h}_t$ is updated as follows:

$$[\boldsymbol{a}^g_t, \boldsymbol{a}^i_t, \boldsymbol{a}^f_t, \boldsymbol{a}^o_t] = \boldsymbol{h}^{cat}_{t-1}\boldsymbol{W}^h + \boldsymbol{b}^h, \tag{5}$$
$$[\boldsymbol{g}_t, \boldsymbol{i}_t, \boldsymbol{f}_t, \boldsymbol{o}_t] = [\phi(\boldsymbol{a}^g_t), \sigma(\boldsymbol{a}^i_t), \sigma(\boldsymbol{a}^f_t), \sigma(\boldsymbol{a}^o_t)], \tag{6}$$
$$\boldsymbol{c}_t = \boldsymbol{g}_t \odot \boldsymbol{i}_t + \boldsymbol{c}_{t-1} \odot \boldsymbol{f}_t, \tag{7}$$
$$\boldsymbol{h}_t = \phi(\boldsymbol{c}_t) \odot \boldsymbol{o}_t, \tag{8}$$

where $\boldsymbol{W}^h \in \mathbb{R}^{(U+M)\times 4M}$ and $\boldsymbol{b}^h \in \mathbb{R}^{4M}$ are parameters, $\boldsymbol{a}^i_t, \boldsymbol{a}^o_t, \boldsymbol{a}^g_t, \boldsymbol{a}^f_t \in \mathbb{R}^M$ are respectively the activations of the input gate $\boldsymbol{i}_t$, the output gate $\boldsymbol{o}_t$, the new content $\boldsymbol{g}_t$, and the forget gate $\boldsymbol{f}_t$, $\boldsymbol{c}_t \in \mathbb{R}^M$ is the updated memory cell, $\sigma(\cdot)$ is the sigmoid function (element-wise), and $\odot$ is the element-wise multiplication. Since LSTM is successful in modelling time series, it is natural to further increase its capacity so that it could be profitably applied to a wider range of tasks.

We consider the width and the depth to compose a network's capacity, where the former measures how much information could be processed in parallel, while the latter measures how many computation steps are required for processing [18]. Whilst using more hidden units in a layer can widen the LSTM, it scales the parameter number quadratically. On the other hand, the Stacked LSTM (*s*LSTM) deepens the LSTM by using multiple layers [19]; however, the runtime scales linearly with the layer number and the input information is likely to be lost when it vertically passes through the LSTM layers (caused by vanishing/exploding gradients).

The goal of this paper is to make the LSTM wider and deeper and meanwhile prevent its parameter number and runtime from growing. To summarize, we have the following contributions:

- We represent RNN hidden states as multidimensional arrays (tensors) to allow more flexible parameter sharing, thereby being able to efficiently widen the network without extra parameters.
- We use the temporal computations of the RNN to absorb its deep computations in order that we can deepen it without extra runtime. We call this novel RNN as the *Tensor RNN (tRNN)*.
- We propose a *memory cell convolution* and apply it to the *t*RNN in order to mitigate gradient vanishing and explosion, obtaining a *Tensor LSTM (tLSTM)*.
- We generalise the *t*LSTM so that it can process not only *non-structured* time series (series of vectors) [20], but also *structured* time series (series of tensors, such as videos).
- We show by experiments that our model is well-suited for various multimedia data modelling tasks, including text generation, text calculation, image classification, and video prediction.

2. Method

2.1. Tensor Representation

From (2), we can see that the parameter number of an RNN scales quadratically with its hidden size. To widen the network while restricting its parameter number, one can use tensor factorisation, where the parameters are represented by multidimensional tensors that could be factorised as low-rank subtensors containing much fewer elements [21–29]. As the hidden state vector would be broadcast when interacting with the parameter tensor, the network is widened implicitly. One can also limit the parameter number of an RNN by spatially sharing a small group of parameters within its hidden state, analogous to the convolutional neural network (CNN) [30,31].

Here, we use parameter sharing to reduce the number of parameters in RNNs. In contrast to tensor factorisation, it provides two benefits: (i) scalability—the size of hidden state would not affect the number of parameters; (ii) separability—we can carefully route the information via the *receptive field* control so that the RNN deep computations can be shifted into its temporal direction (Section 2.2). In addition, the hidden state vectors of RNN are explicitly tensorised, as tensors are more: (i) flexible—we can choose the dimensions for parameter sharing and then just enlarge the sizes of these dimensions so that no more parameters are introduced; (ii) efficient—by using tensors of higher dimensionality, we can widen the network faster when the number of parameters is fixed (Section 2.3).

To ease explanation, let's firstly focus on 2D tensors (matrices). Given a hidden state $\boldsymbol{h}_t \in \mathbb{R}^M$, we tensorise it as $\boldsymbol{H}_t \in \mathbb{R}^{P \times M}$, where P and M are respectively the *tensorised size* and *channel size*. In $\boldsymbol{H}_t$, the 1st dimension is locally-connected for parameter sharing, and the 2nd dimension is fully-connected for global interaction—like in CNN where only the last dimension is fully-connected so that different feature planes (e.g., red/green/blue channels for the input image) can be globally fused. In addition, when comparing $\boldsymbol{H}_t$ with the hidden state in the Stacked RNN (*s*RNN) (as in Figure 1a), P can be thought as the layer number, and M can be thought as each layer's size. To explain our model, we start with 2D tensors, and then demonstrate how to use higher dimensional tensors to enhance the model, and finally show how to extend the model to deal with structured input time series.

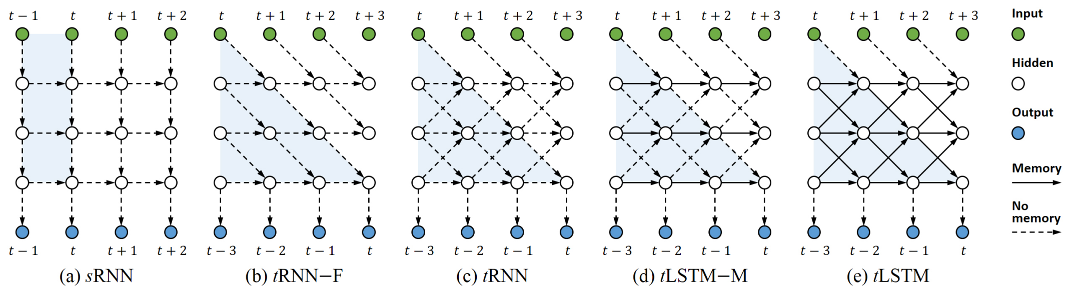

Figure 1. Illustration of the evolution from *s*RNN to *t*LSTM. (**a**) *s*RNN with three layers; (**b**) *t*RNN with no feedback connection (–F); it could be obtained by skewing the *s*RNN shown in (**a**); (**c**) the standard *t*RNN; (**d**) *t*LSTM with no memConv (–M); (**e**) The standard *t*LSTM. For each model, white circles from column 1–4 (from left to right) represent hidden states at time $(t-1)$ to $(t+2)$, respectively. Blue regions represent the output $\boldsymbol{y}_t$'s receptive fields. Note that, in (**b** and **e**), we have delayed the outputs for $L-1=2$ time steps, with a depth $L=3$.

2.2. Deep Computation through Time

As RNN is already *deep* when unfolded in time, we can associate the input $\boldsymbol{x}_t$ with a future (delayed) output to also make the input-to-output computation deep. To achieve this, we should guarantee that the output $\boldsymbol{y}_t$ is *separable*, i.e., it is independent of the future input $\boldsymbol{x}_{t:T}$. Therefore, we first stack $\boldsymbol{x}_t$'s projection on *top* of $\boldsymbol{H}_{t-1}$; then move the input content downwards through the temporal computation, and finally produce $\boldsymbol{y}_t$ from the *bottom* of a future hidden state $\boldsymbol{H}_{t+L-1}$, where $L-1$

denotes the delayed time steps and L denotes the *depth*. Figure 1b shows an example with $L = 3$, which can be thought as a *skewed* *s*RNN that is mentioned in [7,32]. However, in our implementation, the network structure does not need to be changed, and various interactions are also allowed if the output satisfies the separability. For instance, we can use wider local connections or introduce feedback connections (as in Figure 1c) to improve the model (similar to [33]). In addition, we update $\boldsymbol{H}_t$ by convolving it with a learnable kernel so that the parameters can be shared. In doing this, we have increased the input–output mapping complexity (via output delay) and limited the growth of the parameter number (via parameter sharing by convolution).

To define the above described *t*RNN, we denote the concatenated hidden state as $\boldsymbol{H}^{cat}_{t-1} \in \mathbb{R}^{(P+1)\times M}$, and the *location* at a tensor as $p \in \mathbb{Z}_+$. At location p of $\boldsymbol{H}^{cat}_{t-1}$, the channel vector $\boldsymbol{h}^{cat}_{t-1,p} \in \mathbb{R}^M$ satisfies:

$$\boldsymbol{h}^{cat}_{t-1,p} = \begin{cases} \boldsymbol{x}_t \boldsymbol{W}^x + \boldsymbol{b}^x, & \text{if } p = 1 \\ \boldsymbol{h}_{t-1,p-1}, & \text{if } p > 1, \end{cases} \tag{9}$$

where $\boldsymbol{W}^x \in \mathbb{R}^{U\times M}$ and $\boldsymbol{b}^x \in \mathbb{R}^M$. The hidden state $\boldsymbol{H}_t$ is then updated through a convolution:

$$\boldsymbol{A}_t = \boldsymbol{H}^{cat}_{t-1} \circledast \{\boldsymbol{W}^h, \boldsymbol{b}^h\}, \tag{10}$$

$$\boldsymbol{H}_t = \phi(\boldsymbol{A}_t), \tag{11}$$

where $\boldsymbol{W}^h \in \mathbb{R}^{K\times M^i \times M^o}$ and $\boldsymbol{b}^h \in \mathbb{R}^{M^o}$ are the kernel's weight and bias, respectively, with K denoting the kernel size, $M^i = M$ the input channel, and $M^o = M$ the output channel, $\boldsymbol{A}_t \in \mathbb{R}^{P\times M^o}$ denotes the activation of $\boldsymbol{H}_t$, and $\circledast$ denotes the convolution operation (detailed in Appendix A.1). As kernels convolve across different layers of the hidden state, we call the convolution as a *cross-layer convolution*, which allows the interaction among layers (from both top-down and bottom-up). Finally, the channel vector $\boldsymbol{h}_{t+L-1,P} \in \mathbb{R}^M$, located at the bottom of $\boldsymbol{H}_{t+L-1}$, is used to generate $\boldsymbol{y}_t$:

$$\boldsymbol{y}_t = \varphi(\boldsymbol{h}_{t+L-1,P} \boldsymbol{W}^y + \boldsymbol{b}^y), \tag{12}$$

where $\boldsymbol{W}^y \in \mathbb{R}^{M\times V}$ and $\boldsymbol{b}^y \in \mathbb{R}^V$. To ensure $\boldsymbol{y}_t$'s *receptive field* only covers historical inputs $\boldsymbol{x}_{1:t}$ (as in Figure 1c), a constraint among L, P, and K needs to be satisfied:

$$L = \left\lceil \frac{2P}{K - K\%2} \right\rceil, \tag{13}$$

where % is the modulus operator and $\lceil\cdot\rceil$ denotes the ceil operation. Please see in Appendix B for the derivation of (13).

We call the RNN described in (9)–(12) as a *Tensor RNN (tRNN)*, where one can increase the tensorised size P to widen the model, and meanwhile keep the number of parameters fixed (by using convolution). Moreover, different from the *s*RNN with a runtime complexity of $O(TL)$, the runtime complexity of a *t*RNN is broken down to $O(T+L)$, indicating that the runtime would not be significantly affected by T or L.

2.3. Using LSTMs

To capture the long-term dependency across different time steps, the *t*RNN can be straightforwardly extended with LSTM by modifying (10) and (11) as:

$$[\boldsymbol{A}^g_t, \boldsymbol{A}^i_t, \boldsymbol{A}^f_t, \boldsymbol{A}^o_t] = \boldsymbol{H}^{cat}_{t-1} \circledast \{\boldsymbol{W}^h, \boldsymbol{b}^h\}, \tag{14}$$

$$[\boldsymbol{G}_t, \boldsymbol{I}_t, \boldsymbol{F}_t, \boldsymbol{O}_t] = [\phi(\boldsymbol{A}^g_t), \sigma(\boldsymbol{A}^i_t), \sigma(\boldsymbol{A}^f_t), \sigma(\boldsymbol{A}^o_t)], \tag{15}$$

$$\boldsymbol{C}_t = \boldsymbol{G}_t \odot \boldsymbol{I}_t + \boldsymbol{C}_{t-1} \odot \boldsymbol{F}_t, \tag{16}$$

$$\boldsymbol{H}_t = \phi(\boldsymbol{C}_t) \odot \boldsymbol{O}_t, \tag{17}$$

where $\{W^h, b^h\}$ is the kernel with kernel size K, input channel $M^i = M$, and output channel $M^o = 4M$, $A_t^i, A_t^o, A_t^g, A_t^f \in \mathbb{R}^{P \times M}$ are respectively the activations of the input gate I_t, the output gate O_t, the new content G_t, and the forget gate F_t, and $C_t \in \mathbb{R}^{P \times M}$ is the updated memory cell. However, as (16) only gates the previous memory cell C_{t-1} along the temporal direction (as in Figure 1d), when the tensorised size P grows large, the long-term dependency from the input–output direction is likely to be lost.

2.3.1. Memory Cell Convolution

Here, we propose a novel memory cell convolution (memConv) for capturing the long-term dependency from multiple directions, where, like the hidden state, the memory cell could also have a wider receptive field (as in Figure 1e). In addition, the kernel for memConv is generated on the fly and therefore varies with time and location, flexibly controlling the long-term dependency from different directions. Concretely, we define the tensor update for tLSTM as follows:

$$[A_t^g, A_t^i, A_t^f, A_t^o, A_t^q] = H_{t-1}^{cat} \circledast \{W^h, b^h\}, \tag{18}$$

$$[G_t, I_t, F_t, O_t, Q_t] = [\phi(A_t^g), \sigma(A_t^i), \sigma(A_t^f), \sigma(A_t^o), \varsigma(A_t^q)], \tag{19}$$

$$W_t^c(p) = \text{reshape}\left(q_{t,p}, [K, 1, 1]\right), \tag{20}$$

$$C_{t-1}^{conv} = C_{t-1} \circledast W_t^c(p), \tag{21}$$

$$C_t = G_t \odot I_t + C_{t-1}^{conv} \odot F_t, \tag{22}$$

$$H_t = \phi(C_t) \odot O_t, \tag{23}$$

where, unlike (14)–(17), the kernel $\{W^h, b^h\}$ contains additional $\langle K \rangle$ output channels ($\langle \cdot \rangle$ computes the cumulative product of the input variable elements) for generating the activation $A_t^q \in \mathbb{R}^{P \times \langle K \rangle}$ of the dynamic kernel bank $Q_t \in \mathbb{R}^{P \times \langle K \rangle}$, $q_{t,p} \in \mathbb{R}^{\langle K \rangle}$ denotes the vectorised dynamic kernel selected from Q_t's entry p, and $W_t^c(p) \in \mathbb{R}^{K \times 1 \times 1}$ is the dynamic kernel reshaped from $q_{t,p}$ (illustrated in Figure 2a), with a size K and a single input/output channel. Equation (21) defines the memConv (detailed in Appendix A.2), where we use $W_t^c(p)$, the value of which varies with p, to convolve *every channel* of C_{t-1}, producing a convolved memory cell $C_{t-1}^{conv} \in \mathbb{R}^{P \times M}$. Analogous to [34], in (19), a softmax function $\varsigma(\cdot)$ is employed to normalise Q_t along its channel dimension, which can stabilise the memory cell values and thereby mitigate the vanishing and exploding gradients (please check Appendix C for more discussion).

There are many works [22,23,27,35–37] using the concept of dynamically producing the model weights, where [36] also dynamically generates location-dependent convolution kernels for improving the CNN. Unlike these works, we aim to broaden the receptive fields for tLSTM memory cells. Whilst being flexible, fewer parameters are needed for generating the memConv kernel as it can be shared by different channels of the memory cell.

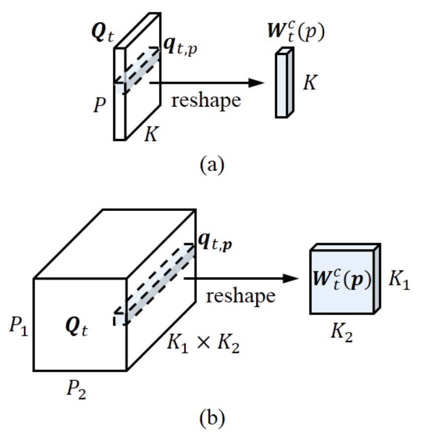

Figure 2. Illustration of how to generate the memConv kernels for 2D (**a**) and 3D (**b**) tensors.

2.3.2. Channel Normalisation

We adapt the recently proposed layer normalisation (LN) [38] to speed-up the training of *t*LSTM. In [38], LN has been observed unsuitable for the CNN where different statistics are possessed by different channel vectors. Similarly, we have found that LN also performs not well for the *t*LSTM, in which low-level information is possessed by channel vectors close to the input and vice versa. Therefore, we propose a *channel normalisation* (CN), which normalises each channel vector independently. The CN operator is defined as:

$$\mathrm{CN}\,(\boldsymbol{Z};\boldsymbol{\Gamma},\boldsymbol{B}) = \widehat{\boldsymbol{Z}} \odot \boldsymbol{\Gamma} + \boldsymbol{B}, \tag{24}$$

where $\boldsymbol{\Gamma}, \boldsymbol{B}, \boldsymbol{Z}, \widehat{\boldsymbol{Z}} \in \mathbb{R}^{P\times M^z}$, $\boldsymbol{\Gamma}$ and $\boldsymbol{B}$ are parameters namely the *gain* and *bias*, respectively, $\boldsymbol{Z}$ is the input tensor, and $\widehat{\boldsymbol{Z}}$ is the normalised tensor. Let $\boldsymbol{z}_{m^z} \in \mathbb{R}^P$ be the m^z-th channel of $\boldsymbol{Z}$, it is normalised element-wisely:

$$\widehat{\boldsymbol{z}}_{m^z} = (\boldsymbol{z}_{m^z} - \boldsymbol{z}^\mu)/\boldsymbol{z}^\sigma, \tag{25}$$

where $\boldsymbol{z}^\mu, \boldsymbol{z}^\sigma \in \mathbb{R}^P$ are respectively the *mean* and the *standard deviation* which are computed along $\boldsymbol{Z}$'s channel dimension, and $\widehat{\boldsymbol{z}}_{m^z} \in \mathbb{R}^P$ denotes the m^z-th channel in $\widehat{\boldsymbol{Z}}$. As the parameter number introduced by CN/LN is quite small in terms of the model parameters, it could be reasonably neglected.

2.3.3. Leveraging Higher-Dimensional Tensors

In (13), we can see that given the kernel size K, the tensorised size P scales linearly w.r.t. the depth L. To widen the *t*LSTM more efficiently, we resort to using higher-dimensional tensors, where the tensor volume can be expanded more rapidly. Based on the *t*LSTM defined in previous sections, we can generalise the tensors from 2D to $(D+1)$-dimensional where $D>1$, resulting in $\boldsymbol{H}_t, \boldsymbol{C}_t \in \mathbb{R}^{P_1\times P_2\times\ldots\times P_D\times M}$ with the tensorised size $P = [P_1, P_2, \ldots, P_D]$. As the hidden states are more than 2D, we instead concatenate $\boldsymbol{x}_t$'s projection to the *corner* of $\boldsymbol{H}_{t-1}$, thereby extending (9) as:

$$\boldsymbol{h}^{cat}_{t-1,\boldsymbol{p}} = \begin{cases} \boldsymbol{x}_t\boldsymbol{W}^x + \boldsymbol{b}^x, & \text{if } p_d = 1 \text{ for } 1 \le d \le D \\ \boldsymbol{h}_{t-1,\boldsymbol{p}-\boldsymbol{1}}, & \text{if } p_d > 1 \text{ for } 1 \le d \le D \\ \boldsymbol{0}, & \text{otherwise,} \end{cases} \tag{26}$$

where the channel vector $\boldsymbol{h}^{cat}_{t-1,\boldsymbol{p}} \in \mathbb{R}^M$ is the entry $\boldsymbol{p} \in \mathbb{Z}^D_+$ of the concatenated hidden state $\boldsymbol{H}^{cat}_{t-1} \in \mathbb{R}^{(P_1+1)\times(P_2+1)\times\ldots\times(P_D+1)\times M}$. Accordingly, the output $\boldsymbol{y}_t$ is generated at the *opposite corner* of $\boldsymbol{H}_{t+L-1}$, thus we modify (12) as:

$$\boldsymbol{y}_t = \varphi(\boldsymbol{h}_{t+L-1,P}\boldsymbol{W}^y + \boldsymbol{b}^y). \tag{27}$$

To update the hidden state, we also tensorise the convolution kernel $\boldsymbol{W}^h$ and $\boldsymbol{W}^c_t(\cdot)$ so that they have a kernel size of $K = [K_1, K_2, \ldots, K_D]$, where $\boldsymbol{W}^c_t(\cdot)$ is still reshaped from the vector (see Figure 2b). In order to make every dimension of P and K meet the constraint (13) with a same L, we set $P_d = P$ and $K_d = K$ for $d = 1, 2, \ldots, D$. For CN, it is still applied to normalise the channel dimension of tensors.

Since D is the additional dimensionality introduced by tensorisation, we call D the *tensorised dimensionality* and denote the resulting *t*LSTM as a D-*t*LSTM. For instance, the untensorised LSTM defined in (5)–(8) is a 0-*t*LSTM, and the *t*LSTM (with 2D tensors) defined in (18)–(23) is an 1-*t*LSTM.

2.4. Handling Structured Inputs

Until now, we have limited our discussion to the case where the input at each time step is a vector, which is *non-structured*. However, as *structured* data (e.g., image time series) also emerges in many multimedia modelling tasks (e.g., video segmentation, motion estimation, and video prediction), it is essential to generalise the model to handle structured inputs.

We use $\boldsymbol{X}_t \in \mathbb{R}^{S_1 \times S_2 \times \ldots \times S_E \times U}$ to denote the structured input at time step t, where $E \in \mathbb{Z}_+$ is the *structure dimensionality* and $S = [S_1, S_2, \ldots, S_E]$ the *structure size*, e.g., when $\boldsymbol{X}_t$ is a 2D image, then $S = [S_1, S_2]$ is the image size (height and width) and U is the image depth (channel). Correspondingly, we have a hidden state $\boldsymbol{H}_t \in \mathbb{R}^{P_1 \times P_2 \times \ldots \times P_D \times S_1 \times S_2 \times \ldots \times S_E \times M}$. In contrast to (26), we define the sub-tensor $\boldsymbol{H}^{cat}_{t-1,\boldsymbol{p}} \in \mathbb{R}^{S_1 \times S_2 \times \ldots \times S_E \times M}$ locating at entry $\boldsymbol{p}$ of $\boldsymbol{H}^{cat}_{t-1}$ as:

$$\boldsymbol{H}^{cat}_{t-1,\boldsymbol{p}} = \begin{cases} \boldsymbol{X}_t \circledast \{\boldsymbol{W}^x, \boldsymbol{b}^x\}, & \text{if } \ p_d = 1 \ \text{ for } \ 1 \le d \le D \\ \boldsymbol{H}_{t-1,\boldsymbol{p}-\boldsymbol{1}}, & \text{if } \ p_d > 1 \ \text{ for } \ 1 \le d \le D \\ \boldsymbol{0}, & \text{otherwise,} \end{cases} \tag{28}$$

where the convolution kernel $\{\boldsymbol{W}^x, \boldsymbol{b}^x\}$ is used for linear projection and is of size $\boldsymbol{1} \in \mathbb{R}^E$, with U input channels and M output channels.

To update the hidden state tensor, the size of convolution kernels $\boldsymbol{W}^h$ and $\boldsymbol{W}^c_t(\cdot)$ becomes $K = [K_1, K_2, \ldots, K_D, K_{D+1}, K_{D+2}, \ldots, K_{D+E}]$, where the first D dimensions, $K_{1:D}$, are related to the tensorised size P, and the succeeding E dimensions, $K_{D+1:D+E}$, are related to the structure size E. This also means that $K_{D+1:D+E}$ are free of the constraint (13).

Finally, we generate the output from the sub-tensor $\boldsymbol{H}_{t+L-1,P} \in \mathbb{R}^{S_1 \times S_2 \times \ldots \times S_E \times M}$. Note that for many tasks such as video prediction, the output usually has the same structure (i.e., a same S) as the input. In this case, the output can be generated by:

$$\boldsymbol{Y}_t = \varphi(\boldsymbol{H}_{t+L-1,P} \circledast \{\boldsymbol{W}^y, \boldsymbol{b}^y\}), \tag{29}$$

where $\boldsymbol{Y}_t \in \mathbb{R}^{S_1 \times S_2 \times \ldots \times S_E \times V}$ is the structured output and $\{\boldsymbol{W}^y, \boldsymbol{b}^y\}$ is the convolution kernel of size $\boldsymbol{1} \in \mathbb{R}^E$, with M input channels and V output channels. In addition, it is straightforward to generate a non-structured output $\boldsymbol{y}_t \in \mathbb{R}^V$ from $\boldsymbol{H}_{t+L-1,P}$, e.g., by using a CNN or a fully-connected network.

3. Related Work

3.1. Convolutional LSTMs

The Convolutional LSTM (*c*LSTM) parallelises the computation of LSTM where at each time step the input is *structured* (as in Figure 3a), such as an array vector [7], a matrix of vectors [39–42], and a tensor of vectors [43,44]. Different from the *c*LSTM, *t*LSTM focuses on increasing the capacity of LSTM where each input can also be *non-structured* (a single vector), and has the following advantages: (i) the convolution in *t*LSTM is performed across different hidden layers, the structure of which can be different from the input structure, integrating information top-down and bottom-up, whereas the convolution in *c*LSTM is only performed within each hidden layer, the structure of which depends on the input structure, thereby falling back to the standard LSTM when each input is a single vector; (ii) by increasing the tensorised size, one can efficiently widen the *t*LSTM without introducing more parameters, whereas to widen the *c*LSTM, either increasing the kernel size or kernel channel can significantly increase the parameter number; (iii) by delaying the output, one can deepen the *t*LSTM with little additional runtime, whereas to deepen the *c*LSTM, increasing the number of hidden layers can significantly increase the runtime; (iv) with the memConv, *t*LSTM can capture the long-term dependency of multiple directions, whereas, *c*LSTM only gates the memory cell along one direction, thereby struggling to capture the long-term dependency of multiple directions.

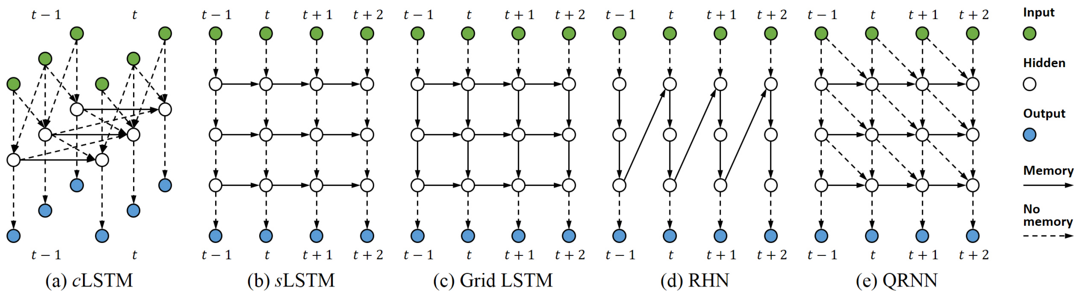

Figure 3. Examples of the related models. (**a**) The *c*LSTM [7] with one layer, where the input at each time step is an array of vectors; (**b**) the *s*LSTM [45] with three layers; (**c**) the Grid LSTM [46] with three layers; (**d**) the recurrent highway network (RHN) [47] with three layers; (**e**) the quasi-recurrent neural network [48] with three layers and a kernel size of 2, where temporal convolution is utilised to parallelise costly computations.

3.2. Deep LSTMs

The Deep LSTM (*d*LSTM) improves the *s*LSTM by further deepening it (as shown in Figure 3b–d). In order to limit the number of parameters as well as make training easy, in [46,47,49,50], another RNN/LSTM is applied to the *depth* direction of *d*LSTMs. However, the runtime is still multiplied by the depth. Though deep computations are accelerated in [32,51], they mainly focus on simple architectures, e.g., *s*LSTMs. Unlike *d*LSTMs, in *t*LSTM, deep computations are performed with little extra runtime, and feedback is enabled by cross-layer convolutions. Furthermore, by utilising higher dimensional tensors, one can increase *t*LSTM's capacity with higher efficiency, while the whole stacked hidden layers in a *d*LSTM only compose a 2D tensor, whose dimensionality is fixed.

3.3. Other Parallelisation Methods

When full input and target sequences are available for training, temporal computations of the time series are parallelised (for instance, by using temporal convolutions like in Figure 3e) in [48,52–56]. Nevertheless, for online inference, since inputs are presented sequentially, these methods can no longer parallelise temporal computations, which will also be blocked by deep computations of each time step, rendering themselves not well-suited for the real-time application which requires a high sampling/output frequencies. On the contrary, as *t*LSTM performs deep computations through temporal computations, it can accelerate both training and online inference for many tasks. This is human-like: when converting the input signal into action, we simultaneously process newly arrived signals in a nonblocking manner. One should also notice that for some tasks (such as autoregressive sequence generation) which take the previous output $\boldsymbol{y}_{t-1}$ as the current input $\boldsymbol{x}_t$, *t*LSTM is unable to parallelise the deep computation for online inference, since additional $L-1$ time steps are required to generate $\boldsymbol{y}_{t-1}$ for each $\boldsymbol{x}_t$.

4. Experiments

To evaluate our *t*LSTM, we experiment on seven challenging multimedia data modelling tasks, and are interested in the following configurations:

- *s*LSTM: We implement the *s*LSTM [45] and share the parameters for different layers. This configuration is served as our baseline.
- *t*LSTM: *t*LSTM with 2D tensors, which is defined in (18)–(23).
- 1-*t*LSTM–M: 1-*t*LSTM with no memConv (–M), i.e., using (14)–(17).
- 1-*t*LSTM–F: 1-*t*LSTM with no feedback connection (–F).
- 2-*t*LSTM: Tensorising 1-*t*LSTM by using 3D tensors, which is explained in Section 2.3.3, with $D=2$.
- 2-*t*LSTM+LN: 2-*t*LSTM with the LN [38].
- 2-*t*LSTM+CN: 2-*t*LSTM with the CN described in Section 2.3.2.

To make different configurations comparable, for sLSTM, we use L and M to represent the layer number and the size of each layer, respectively. Let K be the value of the first D dimensions of the kernel size $\mathcal{K}$, we set $K=2$ for 1-tLSTM–F and $K=3$ for other tLSTM configurations, so that according to (13), we have $L=P$.

To check if tLSTM's performance can be improved without using additional parameters, for each configuration, we use the same amount of parameters and meanwhile increase the tensorised size. We also inspect how the depth can affect the runtime, which is quantified as the averaged milliseconds cost by a single sample's forward and backward passes over a single RNN time step. Then, we evaluate tLSTM's ability by comparing it against the state-of-the-art methods. Finally, we analyze the inner working of tLSTM by visualising its memory cells.

The training objective is to minimise the training loss w.r.t. the parameter $\boldsymbol{\theta}$ (vectorised), i.e.,

$$\min_{\boldsymbol{\theta}} \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} l\left(f(\boldsymbol{x}^d_{n,1:t}; \boldsymbol{\theta}), \boldsymbol{y}^d_{n,t}\right), \tag{30}$$

where N is the number of training sequences, T_n is the length of the n-th training sequence, and $l(\cdot,\cdot)$ is the loss between the prediction and the target. We define $l(\cdot,\cdot)$ as the Mean Squared Error (MSE) for regression problems (our video prediction tasks), and as the cross entropy for classification problems (our other tasks). In all tasks, the training objective is minimised by Adam [57] with a learning rate of 0.001. Forget gate biases are set to 4 for image classification tasks and 1 [58] for others. All models are implemented by Torch7 [59] and accelerated by cuDNN on Tesla K80 GPUs (NVIDIA, Santa Clara, CA, USA).

We only apply CN to the output of the tLSTM hidden state as we have tried different combinations and found this is the most robust way that can always improve the performance for all tasks. With CN, the output of hidden state becomes:

$$\boldsymbol{H}_t = \phi\left(\mathrm{CN}\left(\boldsymbol{C}_t; \boldsymbol{\Gamma}, \boldsymbol{B}\right)\right) \odot \boldsymbol{O}. \tag{31}$$

4.1. Text Generation

The dataset of Hutter Prize Wikipedia [60] is a text file comprising 100 million characters with a vocabulary size of 205, including alphabets, special symbols, and XML markups. This dataset is modelled at character-level, and the goal is generating the next character given all previous ones, e.g.,:

Input : `[[Joachim Vadian]], Swiss humanis,`

Target : `[Joachim Vadian]], Swiss humanist.`

We evaluate all configurations for the depth $L=1,2,3,4$ and use 10 M parameters, so that the channel size M for sLSTM and 1-tLSTM–F is 1120, for other 1-tLSTMs is 901, and for 2-tLSTMs is 522. Bits-per-character (BPC) are used for performance measuring. As in [33], we split the dataset into 90 M/5 M/5 M for training/validation/test. In each iteration, the model is fed with a mini-batch of 100 subsequences of length 50. During the forward pass, the hidden values at the last time step are preserved to initialise the next iteration. We terminate training after 50 epochs.

Figure 4 shows the results. With a larger M, sLSTM and 1-tLSTM–F perform better than other models when $L \leq 2$. When L increases, sLSTM and 1-tLSTM–M boost their performances but get stuck when $L \geq 3$, whereas, with the memConv, the performances of tLSTMs improve, finally surpassing sLSTM and 1-tLSTM–M. With $L = 4$, the performance of 1-tLSTM–F is exceeded by that of 1-tLSTM, which is exceeded by that of 2-tLSTM in turn. Whilst LN benefits 2-tLSTM only when $L \leq 2$, CN consistently benefits 2-tLSTM with different L.

Note that, in each tLSTM configuration, the runtime is nearly constant and largely unaffected by L, while in sLSTM, the runtime is almost proportional to L.

To compare with the state-of-the-art methods, we evaluate a larger 2-*t*LSTM+CN on the test set, where $L = 6$ and $M = 1200$. The results are presented in Table 1. With 50.1 M parameters, our model achieves a BPC of 1.264, and is therefore competitive to the best results [47,50] with similar amount of parameters.

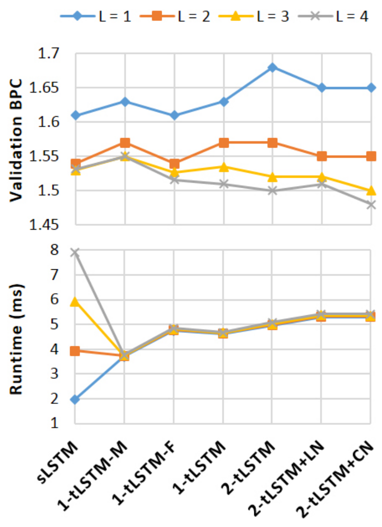

Figure 4. Performance and runtime on Wikipedia.

Table 1. Test BPCs for Wikipedia text generation.

Method	#Parameters	BPC
MI-LSTM [26]	≈17 M	1.44
mLSTM [29]	≈20 M	1.42
HyperLSTM+LN [37]	26.5 M	1.34
HM-LSTM+LN [61]	≈35 M	1.32
Large RHN [47]	≈46 M	1.27
Large FS-LSTM-4 [50]	≈47 M	1.245
2 × Large FS-LSTM-4 [50]	≈94 M	1.198
2-*t*LSTM+CN ($L = 6$, $M = 1200$)	50.1 M	1.264

4.2. Text Calculation

(i) Addition: The goal of this task is adding two integers of 15-digit. The model firstly reads both integers, after which it predicts their sum, both in a sequential manner (i.e., one digit per time step). Following [46], we use the symbol '-' to delimit integers and pad the input and target sequences, e.g.,

$$\begin{aligned} \mathit{Input}: &\quad \texttt{-694104857461284-930283746529103-----------------} \\ \mathit{Target}: &\quad \texttt{--------------------------------1624388603990387-} \end{aligned}$$

(ii) Copy: The copy task is to reproduce 20 random symbols presented as a sequence, where 65 different symbols are used. As in the addition task, the symbol '-' is also used as a delimiter, e.g.,

$$\begin{aligned} \mathit{Input}: &\quad \texttt{-7h@P\}n\$R\&+0\^{}(\#4?w>5C---------------------} \\ \mathit{Target}: &\quad \texttt{---------------------7h@P\}n\$R\&+0\^{}(\#4?w>5C-} \end{aligned}$$

For the *addition* and *copy* tasks, we set M to 400 and 100, respectively, and evaluate each configuration for $L = 1, 4, 7, 10$. The prediction accuracy of symbols are used to measure the performance. Like in [46], for both tasks we randomly generate 5 M training samples and 100 test samples, and set the mini-batch size to 15. Training proceeds for at most one epoch (To simulate the online learning process, we use all training samples only once) and will be terminated if 100% test accuracy is achieved.

Results are shown in Figure 5. In both tasks, the performances of *s*LSTM and 1-*t*LSTM–M degrades with larger L. On the contrary, with L increasing, the performance of 1-*t*LSTM–F continues improving, and can be further boosted by using feedback, tensors of higher dimensionality, and CN, whilst LN improves the performance only when $L = 1$. Note that correct solutions can be found (when achieving 100% test accuracies) in both tasks because of their repetitive nature. From the experiments, we find that in the task of addition, 2-*t*LSTM+CN of $L = 7$ performs the best and solves the task using only 298 K training examples, whilst in the task of copy, 2-*t*LSTM+CN of $L = 10$ outperforms other configurations and copies perfectly using only 54 K training examples. Moreover, different from *s*LSTM, all *t*LSTMs' runtime can be largely independent of L.

On both tasks, the best performing configurations are further compared to the state-of-the-art methods. Table 2 reports the results. Our model solves the tasks of both addition and copy significantly faster (with fewer training examples) than others, being the new state-of-the-art.

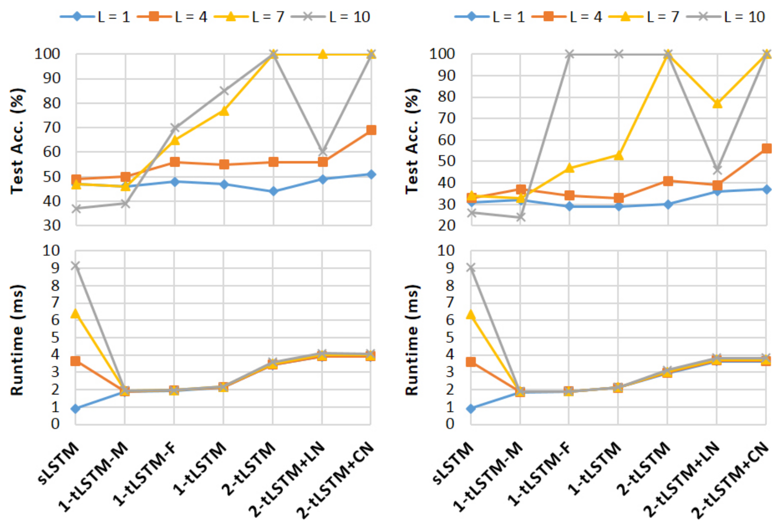

Figure 5. Performance and runtime on text calculation tasks including addition (**left**) and copy (**right**).

Table 2. Test accuracies for addition/copy.

Method	**Addition**		**Copy**	
	#Samples	**Accuracy**	**#Samples**	**Accuracy**
*s*LSTM [45]	5 M	51%	900 K	>50%
Grid LSTM [46]	550 K	>99%	150 K	>99%
2-*t*LSTM+CN ($L = 7$)	298 K	>99%	115 K	>99%
2-*t*LSTM+CN ($L = 10$)	317 K	>99%	54 K	>99%

4.3. Image Classification

The dataset of MNIST [31] comprising 70,000 handwritten digit images sized 28×28, which is divided into 50,000/10,000/10,000 for training/validation/test. For this dataset, there are two tasks:

(i) Sequential MNIST: In this task, the model first sequentially reads the pixels in a scanline order, and then outputs the class of the digit contained in the image [62]. It is a time series task of 784 time steps, where we generate the output from the last time step, thereby requiring to capture very long term temporal dependencies.

(ii) Sequential Permuted MNIST: To make the problem even harder, we generate a permuted MNIST (*p*MNIST) [63] by permuting the original image pixels with a fixed random order so that the long-term dependency can also exist in neighbouring pixels.

In both tasks, we evaluate all configurations with $M = 100$ and $L = 1, 3, 5$. We employ the classification accuracy to measure the model performance. We set the mini-batch size to 50, and use early stopping for training. The training loss is calculated at the last time step.

Figure 6 shows the results. Increasing the depth no longer benefits *s*LSTM and 1-*t*LSTM–M when $L = 5$, while the performance of 1-*t*LSTM can be boosted by a larger depth and tensorisation. However, the performance of 1-*t*LSTM seems not to be affected by removing the feedback connections. In addition, CN always improves 2-*t*LSTM and outperforms LN when $L \geq 3$. With validation accuracies of 99.1% on MNIST and 95.6% on *p*MNIST, 2-*t*LSTM+CN with $L = 5$ outperforms all other configurations in both tasks. In *t*LSTMs, the runtime is little affected by L, and when $L = 5$, all *t*LSTMs runs faster than *s*LSTM.

As presented in Table 3, the best performing configurations are compared against the state-of-the-art methods. On sequential MNIST, 2-*t*LSTM+CN with $L = 3$ achieves 99.2% test accuracy, which is the same as the state-of-the-art one produced by the Dilated GRU [56]. On sequential *p*MNIST, 2-*t*LSTM+CN with $L = 5$ achieves 95.7% test accuracy, approaching the state-of-the-art one of 96.7% which is obtained from the Dilated CNN [54] in [56].

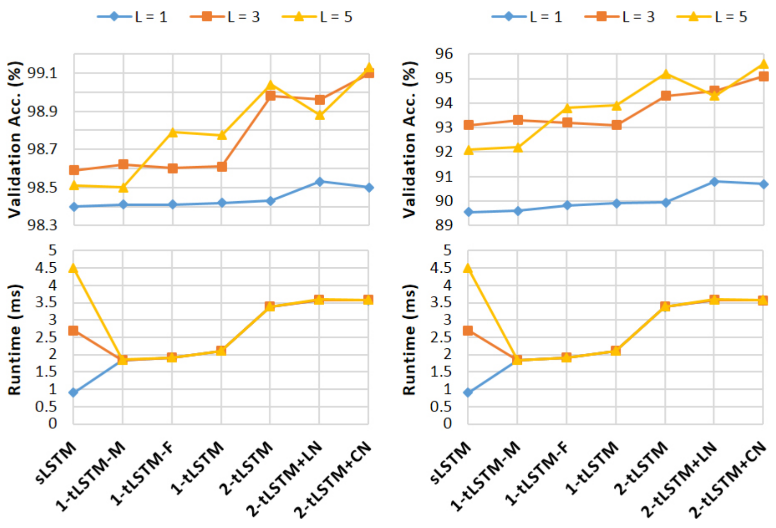

Figure 6. Performance and runtime on sequential MNIST (**left**) and sequential *p*MNIST (**right**).

Table 3. Test accuracies (%) for sequential MNIST/*p*MNIST image classification.

Method	MNIST	*p*MNIST
*i*RNN [62]	97.0	82.0
LSTM [63]	98.2	88.0
*u*RNN [63]	95.1	91.4
Full-capacity *u*RNN [64]	96.9	94.1
*s*TANH [65]	98.1	94.0
BN-LSTM [66]	99.0	95.4
Dilated GRU [56]	99.2	94.6
Dilated CNN [54] in [56]	98.3	96.7
2-*t*LSTM+CN ($L = 3$)	99.2	94.9
2-*t*LSTM+CN ($L = 5$)	99.0	95.7

4.4. Video Prediction

The task of video prediction aims at predicting the future frames of a video given the historical frames. It has a variety of applications such as environment simulation, dataset augmentation, and many computer vision tasks. The main challenge is that the model must capture both the spatial and the temporal relationships among data well. We apply our model to two datasets:

(i) KTH [67]: The dataset consists of 600 real videos with 25 subjects performing six actions (walking, running, jogging, hand-clapping, hand-waving, and boxing). It has been split into a training set (subjects 1–16) and a test set (subjects 17–25), resulting in 383 and 216 sequences, respectively. We resize all frames to 128×128.

(ii) UCF101 [68]: The dataset consists of 13,320 real videos of resolution 320×240 with 101 human actions that could be split into five types (sports, playing musical instruments, human-human interaction, body-motion only, and human-object interaction). It is currently the most challenging dataset of actions. Following [69], we train our models on Sports-1M [70] dataset and test them on UCF-101.

On both tasks, we evaluate all configurations with $L = 1, 3, 5$. To process the structured inputs (i.e., video frames), we modify the original *s*LSTM [45] by replacing each LSTM layer with a Convolutional LSTM [39], where the convolution kernel size is set to $[5, 5]$. We also set the last two dimensions (relevant to image structure) of the convolution kernel size K to 5 for *t*LSTMs. M is set to 100 for KTH and 200 for UCF101. The model performance is measured by three common metrics including MSE, Peak Signal-to-Noise Ratio (PSNR), and Structural Similarity Index Measure (SSIM) [71], where SSIM ranges in $[-1, 1]$ (larger is better). We set the mini-batch size to 16 and employ early stopping for training. All models are trained by observing 10 frames and predicting the next 10 frames.

Figure 7 shows the quantitative results. When L increases, *s*LSTM and 1-*t*LSTM–M improve their performances but get stuck at $L = 5$, while with the memConv, the performances of *t*LSTMs improve and finally exceed both *s*LSTM and 1-*t*LSTM–M. The effects of feedback and tensorisation become significant when L is large. Similar to the finding in [38] that LN is not suitable for normalising the convolution layer for images, the performance of 2-*t*LSTM+LN is even worse than 2-*t*LSTM. However, CN improves 2-*t*LSTM with different L. Unlike *s*LSTM where the runtime increases linearly w.r.t. L, *t*LSTM can keep its runtime largely unchanged when increasing L.

The best performing configuration is compared with the state-of-the-art methods (their source codes are publicly available) on both datasets (see Table 4). 2-*t*LSTM+CN with $L = 5$ outperforms all existing models on KTH w.r.t. all metrics, and on UCF101 w.r.t. MSE and SSIM. Sampled qualitative results produced by 2-*t*LSTM+CN are shown in Figure 8.

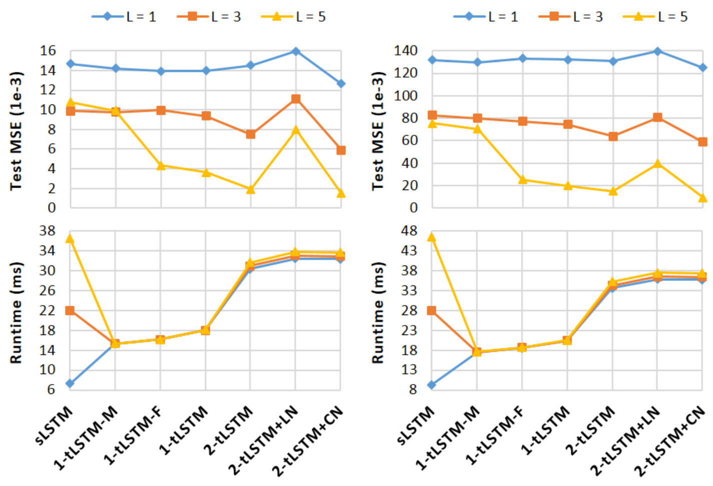

Figure 7. Performance and runtime on KTH (**left**) and UCF101 (**right**).

Table 4. Test performances for KTH/UCF101 video prediction.

Method	**KTH**			**UCF101**		
	MSE↓	**PSNR**↑	**SSIM**↑	**MSE**↓	**PSNR**↑	**SSIM**↑
Composite LSTM [72]	0.01021	20.893	0.77958	0.16342	9.877	0.50363
Beyond MSE [69]	0.00193	28.465	0.88234	0.00987	22.133	0.81254
PredNet [73]	0.00384	27.954	0.90052	0.01672	18.945	0.80827
MCnet [12]	0.00190	30.179	0.91228	0.00979	22.861	0.84392
2-*t*LSTM+CN ($L = 5$)	0.00148	31.053	0.93316	0.00925	22.785	0.85941

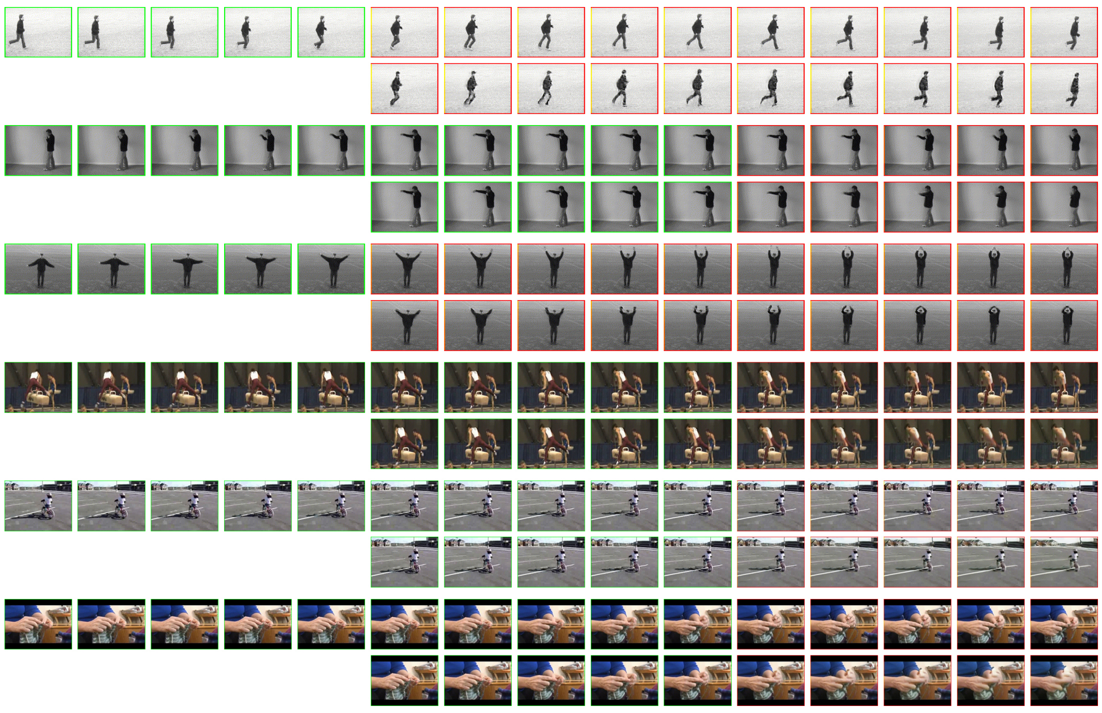

Figure 8. Sampled qualitative results produced by 2-*t*LSTM+CN on KTH (sequences 1 to 3) and UCF101 (sequences 4 to 6). For each sequence, the first row shows the last five input frames (**left**) and the next 10 target frames (**right**), and the second row shows the next 10 predictions. All frames are shown with an aspect ratio of 4:3.

4.5. Analysis

It can been seen from the experiments that one can boost the performance of *t*LSTMs by enlarging the tensorised size or increasing the model depth, whereas almost no extra parameters and runtime are required. The memConv is indispensable to maintain the performance improvement when the network gets wider and deeper. In addition, for tasks with sequential output, feedback connections are useful. In addition, tensorisation or CN can further strengthen the *t*LSTM.

To inspect the inner working of our *t*LSTM, the value of memory cells are visualised to show the information routing. On each task, we run the best performing *t*LSTM with a random sample (here we do not consider video prediction tasks where memory cells of the 2-*t*LSTM are 5D tensors, which are hard to visualise). At each time step, we record the memory cell's channel mean (computed by averaging along the channel dimension, for the 2-*t*LSTM, it has a size of $P \times P$), and visualise its diagonal values from location $\boldsymbol{p}^{in} = [1, 1]$ (close to the input) to $\boldsymbol{p}^{out} = [P, P]$ (close to the output).

As shown in Figure 9, the visualisation result reveals different behaviors of *t*LSTM when handling different tasks:

- Text Generation: If the next character is largely determined by the current input, the input content can be preserved with less modification when it arrives at the output location, and vice versa.
- Addition: Two integers are gradually compressed into the memory and then interact with each other, generating their summation.
- Copy: The model acts as a shift register, continuing to move the input symbols to their output locations.
- Seq. MNIST: The model seems more sensitive to pixel value changes (which represent contours, or the digit topology); it gradually accumulates evidence to generate the final output.
- Seq. *p*MNIST: The model seems more sensitive to high value pixels (which come from the digit); our conjecture is that the permutation has destroyed the digit topology, and thereby made each high value pixel potentially important.

In these tasks, there are also some common phenomena:

- At each time step, different locations of the tensor possess markedly different values, which implies that a tensor of a larger size could encode more content, requiring less effort for compressing.
- The value becomes more and more distinct from the input to output and is shifted along the time axis, which reveals that the model indeed simultaneously performs the deep and temporal computations, with the memory cell carrying the long-term dependency.

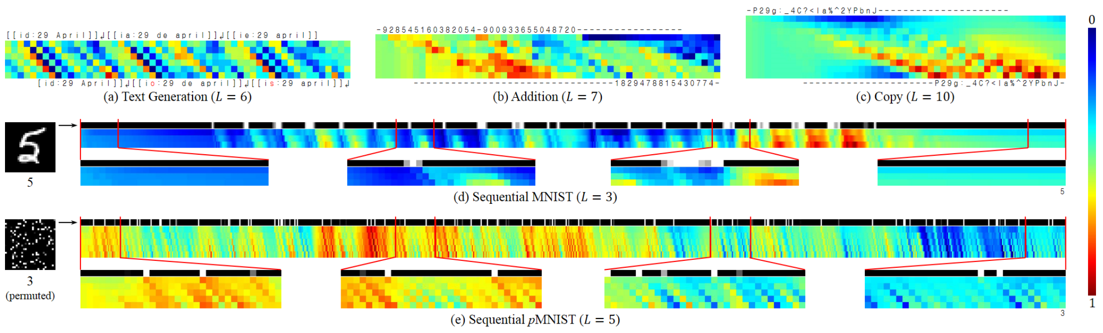

Figure 9. Visualisation of *t*LSTM memory cells' diagonal channel means on different tasks. (**a**) Text Generation ($L = 6$); (**b**) Addition ($L = 7$); (**c**) Copy ($L = 10$); (**d**) Sequential MNIST ($L = 3$); (**e**) Sequential *p*MNIST ($L = 5$). For each colour matrix, the p-th row corresponds to location $[p, p]$, the t-th column corresponds to time step t where $t = 1, 2, \ldots, T + L - 1$ and $L - 1$ denotes the delayed time steps, and all values have been normalised to $[0, 1]$ for visualisation. Note that we have horizontally squeezed the complete sequences in (**d**,**e**) where $T = 784$.

5. Conclusions

In this paper, we have aimed to deal with multimedia modelling tasks. We have introduced the *t*LSTM, where tensors are employed to share parameters and temporal computations are utilised to perform deep computations. The main advantage of our *t*LSTM over other popular methods is that its capacity can be increased with almost no extra parameters and runtime. Another important advantage of the *t*LSTM is that it can handle a variety of challenging multimedia modelling tasks well as shown in our experiments.

For future work, we would like to: (i) investigate more about the effect of higher-dimensional tensors, e.g., try 3- and 4-*t*LSTMs, (ii) try increasing the transition depth for *t*LSTM hidden states (similar to [47]); and (iii) apply *t*LSTMs to more multimedia modelling tasks such as machine translation, image generation, and video segmentation.

Author Contributions: Methodology, Z.H., S.G., L.X., D.L., and H.H.; Software, Z.H. and S.G.; Validation, Z.H. and L.X.; Investigation, Z.H., S.G., and D.L.; Writing—Original Draft Preparation, Z.H.; Writing—Review & Editing, Z.H., S.G., D.L., and H.H.; Visualization, Z.H., S.G., and L.X.; Supervision, D.L. and H.H.; Project Administration, Z.H., D.L., and H.H.

Funding: This work is supported by the Natural Science Foundations of China under Grants 91220301 and 61806134.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. Mathematical Definition for Cross-Layer Convolutions

Appendix A.1. Hidden State Convolution

The hidden state convolution in (10) is defined as:

$$A_{t,p,m^o} = \sum_{k=1}^{K} \left(\sum_{m^i=1}^{M^i} H^{cat}_{t-1,p-\frac{K-1}{2}+k,m^i} \cdot W^h_{k,m^i,m^o} \right) + b^h_{m^o}, \tag{A1}$$

where $m^o \in \{1, 2, \cdots, M^o\}$ and we apply zero padding to maintain the tensorised size.

Appendix A.2. Memory Cell Convolution

The memory cell convolution in (21) is defined as:

$$C^{conv}_{t-1,p,m} = \sum_{k=1}^{K} C_{t-1,p-\frac{K-1}{2}+k,m} \cdot W^c_{t,k,1,1}(p). \tag{A2}$$

To prevent the stored information from being flushed away, $\boldsymbol{C}_{t-1}$ is padded with the replication of its boundary values instead of zeros or input projections.

Appendix B. Derivation for the Constraint of L, P, and K

Here we derive the constraint of L, P, and K that is defined in (13). The kernel center location is sealed in case the kernel size K is not odd. Then, the kernel radius K^r can be calculated by:

$$K^r = \frac{K - K\%2}{2}. \tag{A3}$$

As shown in Figure A1, to guarantee that the receptive field of $\boldsymbol{y}_t$ covers $\boldsymbol{x}_{1:t}$ while not covering $\boldsymbol{x}_{t+1:T}$, the following constraint should be satisfied:

$$\tan \angle \text{AOD} \leqslant \tan \angle \text{BOD} < \tan \angle \text{COD}, \tag{A4}$$

which means:

$$\frac{P}{L} \leqslant \frac{K^r}{1} < \frac{P}{L-1}. \tag{A5}$$

Plugging (A3) into (A5), we get:

$$L = \left\lceil \frac{2P}{K - K\%2} \right\rceil. \tag{A6}$$

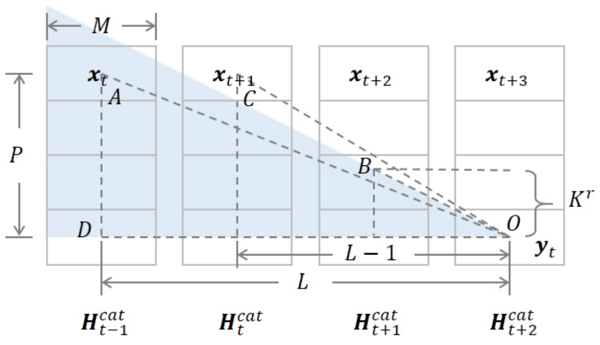

Figure A1. Illustration of calculating the constraint of L, P, and K. Each column is a concatenated hidden state tensor with tensorised size $P+1=4$ and channel size M. The volume of the output receptive field (blue region) is determined by the kernel radius K^r. The output $\boldsymbol{y}_t$ for current time step t is delayed by $L-1=2$ time steps.

Appendix C. MemConv Mitigates the Gradient Vanishing/Explosion

In [34], it has been proved that the *lambda gate*, which is very similar to our memConv kernel, can mitigate the gradient vanishing/explosion (please refer to Theorems 17 and 18 in [34]). The differences between our approach and their *lambda gate* are: (i) we normalise the kernel values though a softmax function, while they normalise the gate values by dividing them with their sum, and (ii) we share the kernel for all channels, while they do not. However, as neither modifications affects the conditions of validity for Theorems 17 and18 in [34], our memConv can also mitigate the gradient vanishing/explosion.

References

1. Socher, R.; Perelygin, A.; Wu, J.; Chuang, J.; Manning, C.D.; Ng, A.; Potts, C. Recursive deep models for semantic compositionality over a sentiment treebank. In Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, Seattle, WA, USA, 18–21 October 2013.
2. Santos, C.D.; Zadrozny, B. Learning character-level representations for part-of-speech tagging. In Proceedings of the 31st International Conference on International Conference on Machine Learning, Beijing, China, 21–26 June 2014.
3. Bahdanau, D.; Cho, K.; Bengio, Y. Neural machine translation by jointly learning to align and translate. In Proceedings of the International Conference on Learning Representations 2015, San Diego, CA, USA, 7–9 May 2015.
4. Iyyer, M.; Boyd-Graber, J.; Claudino, L.; Socher, R.; Daumé, H., III. A neural network for factoid question answering over paragraphs. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar, 25–29 October 2014.
5. Byeon, W.; Breuel, T.M.; Raue, F.; Liwicki, M. Scene labeling with lstm recurrent neural networks. In Proceedings of the 28th IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 7–12 June 2015.
6. Kumar, A.C.; Bhandarkar, S.M.; Prasad, M. Depthnet: A recurrent neural network architecture for monocular depth prediction. In Proceedings of the 2018 Conference on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, UT, USA, 18–22 June 2018.
7. Van den Oord, A.; Kalchbrenner, N.; Kavukcuoglu, K. Pixel Recurrent Neural Networks. In Proceedings of the 33rd International Conference on Machine Learning, New York, NY, USA, 19–24 June 2016.
8. Huang, Y.; Wang, W.; Wang, L. Bidirectional recurrent convolutional networks for multi-frame super-resolution. In Proceedings of the Twenty-ninth Conference on Neural Information Processing Systems, Montreal, QC, Canada, 7–12 December 2015.
9. Milan, A.; Rezatofighi, S.H.; Dick, A.R.; Reid, I.D.; Schindler, K. Online Multi-Target Tracking Using Recurrent Neural Networks. In Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, San Francisco, CA, USA, 4–9 February 2017.

10. Tokmakov, P.; Alahari, K.; Schmid, C. Learning Video Object Segmentation with Visual Memory. In Proceedings of the International Conference on Computer Vision, Venice, Italy, 22–29 October 2017.
11. Ranzato, M.; Szlam, A.; Bruna, J.; Mathieu, M.; Collobert, R.; Chopra, S. Video (language) modeling: A baseline for generative models of natural videos. *arXiv* **2014**, arxiv:1412.6604 .
12. Villegas, R.; Yang, J.; Hong, S.; Lin, X.; Lee, H. Decomposing motion and content for natural video sequence prediction. In Proceedings of the 5th International Conference on Learning Representations, Toulon, France, 24–26 April 2017.
13. Rumelhart, D.E.; Hinton, G.E.; Williams, R.J. Learning representations by back-propagating errors. *Nature* **1986**, *323*, 533–536. [CrossRef]
14. Elman, J.L. Finding structure in time. *Cognit. Sci.* **1990**, *14*, 179–211. [CrossRef]
15. Bengio, Y.; Simard, P.; Frasconi, P. Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Netw.* **1994**, *5*, 157–166. [CrossRef] [PubMed]
16. Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef] [PubMed]
17. Gers, F.A.; Schmidhuber, J.; Cummins, F. Learning to forget: Continual prediction with LSTM. *Neural Comput.* **2000**, *12*, 2451–2471. [CrossRef] [PubMed]
18. Bengio, Y. Learning deep architectures for AI. In *Foundations and Trends® in Machine Learning*; University of California, Berkeley: Berkeley, CA, USA, 2009.
19. Graves, A.; Mohamed, A.R.; Hinton, G. Speech recognition with deep recurrent neural networks. In Proceedings of the 38th International Conference on Acoustics, Speech, and Signal Processing (ICASSP), Vancouver, BC, Canada, 26–30 May 2013.
20. He, Z.; Gao, S.; Xiao, L.; Liu, D.; He, H.; Barber, D. Wider and Deeper, Cheaper and Faster: Tensorized LSTMs for Sequence Learning. In Proceedings of the Thirty-first Conference on Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017.
21. Taylor, G.W.; Hinton, G.E. Factored conditional restricted Boltzmann machines for modeling motion style. In Proceedings of the 26th Annual International Conference on Machine Learning, Montreal, QC, Canada, 14–18 June 2009.
22. Sutskever, I.; Martens, J.; Hinton, G.E. Generating text with recurrent neural networks. In Proceedings of the 28th International Conference on Machine Learning, Bellevue, WA, USA, 28 June–2 July 2011.
23. Denil, M.; Shakibi, B.; Dinh, L.; de Freitas, N.; Ranzato, M.A. Predicting parameters in deep learning. In Proceedings of the Twenty-seventh Conference on Neural Information Processing Systems, Stateline, NV, USA, 5–10 December 2013.
24. Irsoy, O.; Cardie, C. Modeling compositionality with multiplicative recurrent neural networks. In Proceedings of the International Conference on Learning Representations 2015, San Diego, CA, USA, 7–9 May 2015.
25. Novikov, A.; Podoprikhin, D.; Osokin, A.; Vetrov, D.P. Tensorizing neural networks. In Proceedings of the Twenty-ninth Conference on Neural Information Processing Systems, Montreal, QC, Canada, 7–12 December 2015.
26. Wu, Y.; Zhang, S.; Zhang, Y.; Bengio, Y.; Salakhutdinov, R. On Multiplicative Integration with Recurrent Neural Networks. In Proceedings of the Thirtieth Conference on Neural Information Processing Systems, Barcelona, Spain, 5–10 December 2016.
27. Bertinetto, L.; Henriques, J.F.; Valmadre, J.; Torr, P.; Vedaldi, A. Learning feed-forward one-shot learners. In Proceedings of the Thirtieth Conference on Neural Information Processing Systems, Barcelona, Spain, 5–10 December 2016.
28. Garipov, T.; Podoprikhin, D.; Novikov, A.; Vetrov, D. Ultimate tensorization: Compressing convolutional and FC layers alike. In Proceedings of the Thirtieth Conference on Neural Information Processing Systems, Barcelona, Spain, 5–10 December 2016.
29. Krause, B.; Lu, L.; Murray, I.; Renals, S. Multiplicative LSTM for sequence modelling. In Proceedings of the 5th International Conference on Learning Representations, Toulon, France, 24–26 April 2017.
30. LeCun, Y.; Boser, B.; Denker, J.S.; Henderson, D.; Howard, R.E.; Hubbard, W.; Jackel, L.D. Backpropagation applied to handwritten zip code recognition. *Neural Comput.* **1989**, *1*, 541–551. [CrossRef]
31. LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]

32. Appleyard, J.; Kocisky, T.; Blunsom, P. Optimizing Performance of Recurrent Neural Networks on GPUs. *arXiv* **2016**, arxiv:1604.01946.
33. Chung, J.; Gulcehre, C.; Cho, K.; Bengio, Y. Gated Feedback Recurrent Neural Networks. In Proceedings of the 32nd International Conference on Machine Learning, Lille, France, 6–11 July 2015.
34. Leifert, G.; Strauß, T.; Grüning, T.; Wustlich, W.; Labahn, R. Cells in multidimensional recurrent neural networks. *J. Mach. Learn. Res.* **2016**, *17*, 3313–3349.
35. Schmidhuber, J. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Comput.* **1992**, *4*, 131–139. [CrossRef]
36. De Brabandere, B.; Jia, X.; Tuytelaars, T.; Van Gool, L. Dynamic filter networks. In Proceedings of the Thirtieth Conference on Neural Information Processing Systems, Barcelona, Spain, 5–10 December 2016.
37. Ha, D.; Dai, A.; Le, Q.V. HyperNetworks. In Proceedings of the 5th International Conference on Learning Representations, Toulon, France, 24–26 April 2017.
38. Ba, J.L.; Kiros, J.R.; Hinton, G.E. Layer Normalization. *arXiv* **2016**, arxiv:1607.06450.
39. Xingjian, S.; Chen, Z.; Wang, H.; Yeung, D.Y.; Wong, W.K.; Woo, W.C. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In Proceedings of the Twenty-ninth Conference on Neural Information Processing Systems, Montreal, QC, Canada, 7–12 December 2015.
40. Romera-Paredes, B.; Torr, P.H.S. Recurrent instance segmentation. In Proceedings of the 14th European Conference on Computer Vision, Amsterdam, The Netherlands, 11–14 October 2016.
41. Patraucean, V.; Handa, A.; Cipolla, R. Spatio-temporal video autoencoder with differentiable memory. In Proceedings of the International Conference on Learning Representations, San Juan, Puerto Rico, 2–4 May 2016.
42. Wu, L.; Shen, C.; Hengel, A.V.D. Deep Recurrent Convolutional Networks for Video-based Person Re-identification: An End-to-End Approach. *arXiv* **2016**, arxiv:1606.01609.
43. Stollenga, M.F.; Byeon, W.; Liwicki, M.; Schmidhuber, J. Parallel multi-dimensional LSTM, with application to fast biomedical volumetric image segmentation. In Proceedings of the Twenty-ninth Conference on Neural Information Processing Systems, Montreal, QC, Canada, 7–12 December 2015.
44. Chen, J.; Yang, L.; Zhang, Y.; Alber, M.; Chen, D.Z. Combining Fully Convolutional and Recurrent Neural Networks for 3D Biomedical Image Segmentation. In Proceedings of the Thirtieth Conference on Neural Information Processing Systems, Barcelona, Spain, 5–10 December 2016.
45. Graves, A. Generating sequences with recurrent neural networks. *arXiv* **2013**, arxiv:1308.0850.
46. Kalchbrenner, N.; Danihelka, I.; Graves, A. Grid long short-term memory. In Proceedings of the International Conference on Learning Representations (ICLR), San Juan, Puerto Rico, 2–4 May 2016.
47. Zilly, J.G.; Srivastava, R.K.; Koutník, J.; Schmidhuber, J. Recurrent Highway Networks. In Proceedings of the 34th International Conference on Machine Learning (ICML 2017), Sydney, Australia, 6–11 August 2017.
48. Bradbury, J.; Merity, S.; Xiong, C.; Socher, R. Quasi-recurrent neural networks. In Proceedings of the 5th International Conference on Learning Representations, Toulon, France, 24–26 April 2017.
49. Graves, A. Adaptive Computation Time for Recurrent Neural Networks. *arXiv* **2016**, arxiv:1603.08983.
50. Mujika, A.; Meier, F.; Steger, A. Fast-Slow Recurrent Neural Networks. In Proceedings of the Thirty-first Conference on Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017.
51. Diamos, G.; Sengupta, S.; Catanzaro, B.; Chrzanowski, M.; Coates, A.; Elsen, E.; Engel, J.; Hannun, A.; Satheesh, S. Persistent RNNs: Stashing Recurrent Weights On-Chip. In Proceedings of the 33rd International Conference on Machine Learning (ICML 2016), New York, NY, USA, 19–24 June 2016.
52. Kaiser, Ł.; Sutskever, I. Neural gpus learn algorithms. In Proceedings of the International Conference on Learning Representations (ICLR), San Juan, Puerto Rico, 2–4 May 2016.
53. Kaiser, Ł.; Bengio, S. Can Active Memory Replace Attention? In Proceedings of the Thirtieth Conference on Neural Information Processing Systems, Barcelona, Spain, 5–10 December 2016.
54. Van den Oord, A.; Dieleman, S.; Zen, H.; Simonyan, K.; Vinyals, O.; Graves, A.; Kalchbrenner, N.; Senior, A.; Kavukcuoglu, K. Wavenet: A generative model for raw audio. *arXiv* **2016**, arxiv:1609.03499.
55. Lei, T.; Zhang, Y. Training RNNs as Fast as CNNs. *arXiv* **2017**, arxiv:1709.02755.
56. Chang, S.; Zhang, Y.; Han, W.; Yu, M.; Guo, X.; Tan, W.; Cui, X.; Witbrock, M.; Hasegawa-Johnson, M.; Huang, T. Dilated Recurrent Neural Networks. In Proceedings of the Thirty-first Conference on Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017.

57. Kingma, D.; Ba, J. Adam: A method for stochastic optimization. In Proceedings of the International Conference on Learning Representations 2015, San Diego, CA, USA, 7–9 May 2015.
58. Jozefowicz, R.; Zaremba, W.; Sutskever, I. An Empirical Exploration of Recurrent Network Architectures. In Proceedings of the 32nd International Conference on Machine Learning (ICML 2015), Lille, France, 6–11 July 2015.
59. Collobert, R.; Kavukcuoglu, K.; Farabet, C. Torch7: A matlab-like environment for machine learning. In Proceedings of the Twenty-fifth Conference on Neural Information Processing Systems, Sierra Nevada, Spain, 12–17 December 2011.
60. Hutter, M. The Human Knowledge Compression Contest. 2012. Available online: http://prize.hutter1.net (accessed on 13 July 2018).
61. Chung, J.; Ahn, S.; Bengio, Y. Hierarchical multiscale recurrent neural networks. In Proceedings of the 5th International Conference on Learning Representations, Toulon, France, 24–26 April 2017.
62. Le, Q.V.; Jaitly, N.; Hinton, G.E. A simple way to initialize recurrent networks of rectified linear units. *arXiv* **2015**, arxiv:1504.00941.
63. Arjovsky, M.; Shah, A.; Bengio, Y. Unitary Evolution Recurrent Neural Networks. In Proceedings of the 33rd International Conference on Machine Learning (ICML 2016), New York, NY, USA, 19–24 June 2016.
64. Wisdom, S.; Powers, T.; Hershey, J.; Le Roux, J.; Atlas, L. Full-capacity unitary recurrent neural networks. In Proceedings of the Thirtieth Conference on Neural Information Processing Systems, Barcelona, Spain, 5–10 December 2016.
65. Zhang, S.; Wu, Y.; Che, T.; Lin, Z.; Memisevic, R.; Salakhutdinov, R.R.; Bengio, Y. Architectural Complexity Measures of Recurrent Neural Networks. In Proceedings of the Thirtieth Conference on Neural Information Processing Systems, Barcelona, Spain, 5–10 December 2016.
66. Cooijmans, T.; Ballas, N.; Laurent, C.; Courville, A. Recurrent Batch Normalization. In Proceedings of the 5th International Conference on Learning Representations, Toulon, France, 24–26 April 2017.
67. Schuldt, C.; Laptev, I.; Caputo, B. Recognizing human actions: A local SVM approach. In Proceedings of the 17th International Conference on Pattern Recognition, Cambridge, UK, 23–26 August 2004.
68. Soomro, K.; Zamir, A.R.; Shah, M. UCF101: A dataset of 101 human actions classes from videos in the wild. *arXiv* **2012**, arxiv:1212.0402.
69. Mathieu, M.; Couprie, C.; LeCun, Y. Deep multi-scale video prediction beyond mean square error. In Proceedings of the International Conference on Learning Representations (ICLR), San Juan, Puerto Rico, 2–4 May 2016.
70. Karpathy, A.; Toderici, G.; Shetty, S.; Leung, T.; Sukthankar, R.; Fei-Fei, L. Large-scale video classification with convolutional neural networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Columbus, OH, USA, 23–28 June 2014.
71. Wang, Z.; Bovik, A.C.; Sheikh, H.R.; Simoncelli, E.P. Image quality assessment: From error visibility to structural similarity. *IEEE Trans. Image Process.* **2004**, *13*, 600–612. [CrossRef] [PubMed]
72. Srivastava, N.; Mansimov, E.; Salakhudinov, R. Unsupervised learning of video representations using lstms. In Proceedings of the 32nd International Conference on Machine Learning (ICML 2015), Lille, France, 6–11 July 2015.
73. Lotter, W.; Kreiman, G.; Cox, D. Deep predictive coding networks for video prediction and unsupervised learning. In Proceedings of the 5th International Conference on Learning Representations, Toulon, France, 24–26 April 2017.