Compact Autoregressive Network

Di Wang,[1] **Feiqing Huang,**[1] **Jingyu Zhao,**[1] **Guodong Li,**[1*] **Guangjian Tian**[2]
[1]Department of Statistics and Actuarial Science, The University of Hong Kong, Hong Kong, China
[2]Huawei Noah's Ark Lab, Hong Kong, China
{diwang, amieehuang, gladys17, gdli}@hku.hk, Tian.Guangjian@huawei.com

Abstract

Autoregressive networks can achieve promising performance in many sequence modeling tasks with short-range dependence. However, when handling high-dimensional inputs and outputs, the massive amount of parameters in the network leads to expensive computational cost and low learning efficiency. The problem can be alleviated slightly by introducing one more narrow hidden layer to the network, but the sample size required to achieve a certain training error is still substantial. To address this challenge, we rearrange the weight matrices of a linear autoregressive network into a tensor form, and then make use of Tucker decomposition to represent low-rank structures. This leads to a novel compact autoregressive network, called Tucker AutoRegressive (TAR) net. Interestingly, the TAR net can be applied to sequences with long-range dependence since the dimension along the sequential order is reduced. Theoretical studies show that the TAR net improves the learning efficiency, and requires much fewer samples for model training. Experiments on synthetic and real-world datasets demonstrate the promising performance of the proposed compact network.

Introduction

Sequence modeling has been used to address a broad range of applications, including macroeconomic time series forecasting, financial asset management, speech recognition and machine translation. Recurrent neural networks (RNN) and their variants, such as Long-Short Term Memory (Hochreiter and Schmidhuber 1997) and Gated Recurrent Unit (Cho et al. 2014), are commonly used as the default architecture or even the synonym of sequence modeling by deep learning practitioners (Goodfellow, Bengio, and Courville 2016). In the meanwhile, especially for high-dimensional time series, we may also consider the autoregressive modeling or multi-task learning,

$$\widehat{\boldsymbol{y}}_t = f(\boldsymbol{y}_{t-1}, \boldsymbol{y}_{t-2}, \ldots, \boldsymbol{y}_{t-P}), \tag{1}$$

where the output $\widehat{\boldsymbol{y}}_t$ and each input $\boldsymbol{y}_{t-i}$ are N-dimensional, and the lag P can be very large for accomodating sequential dependence. Some non-recurrent feed-forward networks with convolutional or other specific architectures have been proposed recently for sequence modeling, and are shown to have state-of-the-art accuracy. For example, some autoregressive networks, such as PixelCNN (Van den Oord et al. 2016b) and WaveNet (Van den Oord et al. 2016a) for image and audio sequence modeling, are compelling alternatives to the recurrent networks.

This paper aims at the autoregressive model (1) with a large number of sequences. This problem can be implemented by a fully connected network with NP inputs and N outputs. The number of weights will be very large when the number of sequences N is large, and it will be much larger if the data have long-range sequential dependence. This will lead to excessive computational burden and low learning efficiency. Recently, Du et al. (2018) showed that the sample complexity in training a convolutional neural network (CNN) is directly related to network complexity, which indicates that compact models are highly desirable when available samples have limited sizes.

To reduce the redundancy of parameters in neural networks, many low-rank based approaches have been investigated. One is to reparametrize the model, and then to modify the network architecture accordingly. Modification of architectures for model compression can be found from the early history of neural networks (Fontaine, Ris, and Boite 1997; Grézl et al. 2007). For example, a bottleneck layer with a smaller number of units can be imposed to constrain the amount of information traversing the network, and to force a compact representation of the original inputs in a multilayer perceptron (MLP) or an autoencoder (Hinton and Salakhutdinov 2006). The bottleneck architecture is equivalent to a fully connected network with a low-rank constraint on the weight matrix in a linear network.

Another approach is to constrain the rank of parameter matrices directly. For instance, Denil et al. (2013) demonstrated significant redundancy in large CNNs, and proposed a low-rank structure of weight matrices to reduce it. If we treat weights in a layer as a multi-dimensional tensor, tensor decomposition methods can then be employed to represent the low-rank structure, and hence compress the network. Among these works, Lebedev et al. (2014) applied the CP decomposition for the 4D kernel of a single convo-

*Corresponding author.

lution layer to speed up CNN, and Jaderberg, Vedaldi, and Zisserman (2014) proposed to construct a low-rank basis of filters to exploit cross-channel or filter redundancy. Kim et al. (2016) utilized the Tucker decomposition to compress the whole network by decomposing convolution and fully connected layers. The tensor train format was employed in Novikov et al. (2015) to reduce the parameters in fully connected layers. Several tensor decomposition methods were also applied to compress RNNs (Tjandra, Sakti, and Nakamura 2018; Ye et al. 2018; Pan et al. 2019). In spite of the empirical success of low-rank matrix and tensor approaches, theoretical studies for learning efficiency are still limited.

A fully connected autoregressive network for (1) will have N^2P weights, and it will reduce to $Nr + NPr$ for an MLP with one hidden layer and r hidden units. The bottleneck architecture still has too many parameters and, more importantly, it does not attempt to explore the possible compact structure along the sequential order. We first simplify the autoregressive network into a touchable framework, by rearranging all weights into a tensor. We further apply Tucker decomposition to introduce a low-dimensional structure and translate it into a compact autoregressive network, called Tucker AutoRegressive (TAR) net. It is a special compact CNN with interpretable architecture. Different from the original autoregressive network, the TAR net is more suitable for sequences with long-range dependence since the dimension along the sequential order is reduced.

There are three main contributions in this paper:

1. We innovatively tensorize weight matrices to create an extra dimension to account for the sequential order and apply tensor decomposition to exploit the low-dimensional structure along all directions. Therefore, the resulting network can handle sequences with long-range dependence.

2. We provide theoretical guidance on the sample complexity of the proposed network. Our problem is more challenging than other supervised learning problems owing to the strong dependency on sequential samples and the multi-task learning nature. Moreover, our sample complexity analysis can be extended to other feed-forward networks.

3. The proposed compact autoregressive network can flexibly accommodate nonlinear mappings, and offer physical interpretations by extracting explainable latent features.

Linear Autoregressive Network

This section demonstrates the methodology by considering a linear version of (1), and theoretically studies the sample complexity of the corresponding network.

Preliminaries and Background

Notation We follow the notations in Kolda and Bader (2009) to denote vectors by lowercase boldface letters, e.g. $\boldsymbol{a}$; matrices by capital boldface letters, e.g. $\boldsymbol{A}$; tensors of order 3 or higher by Euler script boldface letters, e.g. $\boldsymbol{\mathcal{A}}$. For a generic d^{th}-order tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{p_1 \times \cdots \times p_d}$, denote its elements by $\boldsymbol{\mathcal{A}}(i_1, i_2, \ldots, i_d)$ and unfolding of $\boldsymbol{\mathcal{A}}$ along the n-mode by $\boldsymbol{\mathcal{A}}_{(n)}$, where the columns of $\boldsymbol{\mathcal{A}}_{(n)}$ are the n-mode vectors of $\boldsymbol{\mathcal{A}}$, for $n = 1, 2, \ldots, d$. The vectorization operation is denoted by $\text{vec}(\cdot)$. The inner product of two tensors $\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}} \in \mathbb{R}^{p_1 \times \cdots \times p_d}$ is defined as $\langle \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}} \rangle = \sum_{i_1} \cdots \sum_{i_d} \boldsymbol{\mathcal{A}}(i_1, \ldots, i_d) \boldsymbol{\mathcal{B}}(i_1, \ldots, i_d)$. The Frobenius norm of a tensor $\boldsymbol{\mathcal{A}}$ is defined as $\|\boldsymbol{\mathcal{A}}\|_{\text{F}} = \sqrt{\langle \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{A}} \rangle}$. The mode-$n$ multiplication $\times_n$ of a tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{p_1 \times \cdots \times p_d}$ and a matrix $\boldsymbol{B} \in \mathbb{R}^{q_n \times p_n}$ is defined as

$$
\begin{aligned}
&(\boldsymbol{\mathcal{A}} \times_n \boldsymbol{B})(i_1, \ldots, j_n, \ldots, i_d) \\
&= \sum_{i_n=1}^{p_n} \boldsymbol{\mathcal{A}}(i_1, \ldots, i_n, \ldots, i_d) \boldsymbol{B}(j_n, i_n),
\end{aligned}
$$

for $n = 1, \ldots, d$, respectively. For a generic symmetric matrix $\boldsymbol{A}$, $\lambda_{\max}(\boldsymbol{A})$ and $\lambda_{\min}(\boldsymbol{A})$ represent its largest and smallest eigenvalues, respectively.

Tucker decomposition The Tucker ranks of $\boldsymbol{\mathcal{A}}$ are defined as the matrix ranks of the unfoldings of $\boldsymbol{\mathcal{A}}$ along all modes, namely $\text{rank}_i(\boldsymbol{\mathcal{A}}) = \text{rank}(\boldsymbol{\mathcal{A}}_{(i)})$, $i = 1, \ldots, d$. If the Tucker ranks of $\boldsymbol{\mathcal{A}}$ are $r_1, \ldots, r_d$, where $1 \le r_i \le p_i$, there exist a tensor $\boldsymbol{\mathcal{G}} \in \mathbb{R}^{r_1 \times \cdots \times r_d}$ and matrices $\boldsymbol{U}_i \in \mathbb{R}^{p_i \times r_i}$, such that

$$
\boldsymbol{\mathcal{A}} = \boldsymbol{\mathcal{G}} \times_1 \boldsymbol{U}_1 \times_2 \boldsymbol{U}_2 \cdots \times_d \boldsymbol{U}_d, \tag{2}
$$

which is known as Tucker decomposition (Tucker 1966), and denoted by $\boldsymbol{\mathcal{A}} = [\![\boldsymbol{\mathcal{G}}; \boldsymbol{U}_1, \boldsymbol{U}_2, \ldots, \boldsymbol{U}_d]\!]$. With the Tucker decomposition (2), the n-mode matricization of $\boldsymbol{\mathcal{A}}$ can be written as

$$
\boldsymbol{\mathcal{A}}_{(n)} = \boldsymbol{U}_n \boldsymbol{\mathcal{G}}_{(n)} (\boldsymbol{U}_d \otimes \cdots \otimes \boldsymbol{U}_{n+1} \otimes \boldsymbol{U}_{n-1} \otimes \cdots \otimes \boldsymbol{U}_1)^\top,
$$

where $\otimes$ denotes the Kronecker product for matrices.

Linear Autoregressive Network

Consider a linear autoregressive network,

$$
\boldsymbol{h}_t = \boldsymbol{A}_1 \boldsymbol{y}_{t-1} + \boldsymbol{A}_2 \boldsymbol{y}_{t-2} + \cdots + \boldsymbol{A}_{t-P} \boldsymbol{y}_{t-P} + \boldsymbol{b},
$$

where $\boldsymbol{h}_t = \widehat{\boldsymbol{y}}_t$ is the output, $\boldsymbol{A}_i$s are $N \times N$ weight matrices, and $\boldsymbol{b}$ is the bias vector. Let $\boldsymbol{x}_t = (\boldsymbol{y}_{t-1}^\top, \ldots, \boldsymbol{y}_{t-P}^\top)^\top$ be the NP-dimensional inputs. We can rewrite it into a fully connected network,

$$
\boldsymbol{h}_t = \boldsymbol{W} \boldsymbol{x}_t + \boldsymbol{b}, \tag{3}
$$

for $t = 1, \ldots, T$, where $\boldsymbol{W} = (\boldsymbol{A}_1, ..., \boldsymbol{A}_P) \in \mathbb{R}^{N \times NP}$ is the weight matrix. Note that T denotes the effective sample size, which is the number of samples for training. In other words, the total length of the sequential data is $T + P$.

To reduce the dimension of $\boldsymbol{W}$, a common strategy is to constrain the rank of $\boldsymbol{W}$ to be r, which is much smaller than N. The low-rank weight matrix $\boldsymbol{W}$ can be factorized as $\boldsymbol{W} = \boldsymbol{A}\boldsymbol{B}$, where $\boldsymbol{A}$ is a $N \times r$ matrix and $\boldsymbol{B}$ is a $r \times NP$ matrix, and the network can be transformed into

$$
\boldsymbol{h}_t = \boldsymbol{A}\boldsymbol{B}\boldsymbol{x}_t + \boldsymbol{b}. \tag{4}
$$

The matrix factorization reduces the number of parameters in $\boldsymbol{W}$ from N^2P to $Nr + NPr$. However, if both N and P are large, the weight matrix $\boldsymbol{B}$ is still of large size.

We alternatively rearrange the weight matrices $\boldsymbol{A}_i$s into a 3^{rd}-order tensor $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{N \times N \times P}$ such that $\boldsymbol{\mathcal{W}}_{(1)} = \boldsymbol{W}$; see Figure 1 for the illustration. The Tucker decomposition can then be applied to reduce the dimension from three modes

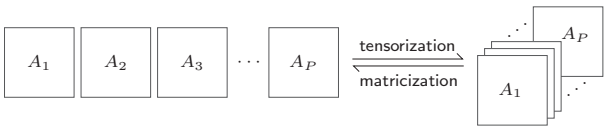

Figure 1: Rearranging P weight matrices of a linear autoregressive network into a tensor.

simultaneously. If the low-Tucker-rank structure is applied on $\mathcal{W}$ with ranks r_1, r_2, r_3, the network becomes

$$\boldsymbol{h}_t = \boldsymbol{U}_1 \mathcal{G}_{(1)} (\boldsymbol{U}_3 \otimes \boldsymbol{U}_2)^\top \boldsymbol{x}_t + \boldsymbol{b}, \tag{5}$$

by Tucker decomposition $\mathcal{W} = [\![\mathcal{G}; \boldsymbol{U}_1, \boldsymbol{U}_2, \boldsymbol{U}_3]\!]$. The Tucker decomposition further reduces the dimension from the other two modes of low-rank structure in (4), while the low-rankness of $\boldsymbol{W}$ only considers the low-dimensional structure on the 1-mode of $\mathcal{W}$ but ignores the possible compact structure on the other two modes.

We train the network based on the squared loss. For simplicity, each sequence is subtracted by its mean, so the bias vector $\boldsymbol{b}$ can be disregarded. The weight matrix or tensor in (3), (4) and (5) can be trained, respectively, by minimizing the following ordinary least squares (OLS), low-rank (LR) and low-Tucker-rank (LTR) objective functions,

$$\widehat{\boldsymbol{W}}_{\text{OLS}} = \arg\min_{\boldsymbol{W}} \frac{1}{T} \sum_{t=1}^{T} \|\boldsymbol{y}_t - \boldsymbol{W}\boldsymbol{x}_t\|_2^2,$$

$$\widehat{\boldsymbol{W}}_{\text{LR}} = \widehat{\boldsymbol{A}}\widehat{\boldsymbol{B}} = \arg\min_{\boldsymbol{A},\boldsymbol{B}} \frac{1}{T} \sum_{t=1}^{T} \|\boldsymbol{y}_t - \boldsymbol{A}\boldsymbol{B}\boldsymbol{x}_t\|_2^2,$$

$$\widehat{\mathcal{W}}_{\text{LTR}} = [\![\widehat{\mathcal{G}}; \widehat{\boldsymbol{U}}_1, \widehat{\boldsymbol{U}}_2, \widehat{\boldsymbol{U}}_3]\!]$$

$$= \arg\min_{\mathcal{G},\boldsymbol{U}_1,\boldsymbol{U}_2,\boldsymbol{U}_3} \frac{1}{T} \sum_{t=1}^{T} \|\boldsymbol{y}_t - \boldsymbol{U}_1 \mathcal{G}_{(1)} (\boldsymbol{U}_3 \otimes \boldsymbol{U}_2)^\top \boldsymbol{x}_t\|_2^2.$$

These three minimizers are called OLS, LR and LTR estimators of weights, respectively.

The matrix factorization or tensor Tucker decomposition is not unique. Conventionally, orthogonal constraints can be applied to these components to address the uniqueness issue. However, we do not impose any constraints on the components to simplify the optimization and mainly focus on the whole weight matrix or tensor instead of its decomposition.

Sample Complexity Analysis

The sample complexity of a neural network is defined as the training sample size requirement to obtain a certain training error with a high probability, and is a reasonable measure of learning efficiency. We conduct a sample complexity analysis for the three estimators, $\widehat{\boldsymbol{W}}_{\text{OLS}}$, $\widehat{\boldsymbol{W}}_{\text{LR}}$ and $\widehat{\mathcal{W}}_{\text{LTR}}$, under the high-dimensional setting by allowing both N and P to grow with the sample size T at arbitrary rates.

We further assume that the sequence $\{\boldsymbol{y}_t\}$ is generated from a linear autoregressive process with additive noises,

$$\boldsymbol{y}_t = \boldsymbol{A}_1 \boldsymbol{y}_{t-1} + \boldsymbol{A}_2 \boldsymbol{y}_{t-2} + \cdots + \boldsymbol{A}_{t-P} \boldsymbol{y}_{t-P} + \boldsymbol{e}_t. \tag{6}$$

Denote by $\boldsymbol{W}_0 = (\boldsymbol{A}_1, \boldsymbol{A}_2, \ldots, \boldsymbol{A}_P)$ the true parameters in (6) and by $\mathcal{W}_0$ the corresponding folded tensor. We assume that $\mathcal{W}_0$ has Tucker ranks r_1, r_2 and r_3, and require the following conditions to hold.

Condition 1. *All roots of matrix polynomial $|\boldsymbol{I}_N - \boldsymbol{A}_1 z - \cdots - \boldsymbol{A}_P z^P| = 0$ are outside unit circle.*

Condition 2. *The errors $\{\boldsymbol{e}_t\}$ is a sequence of independent Gaussian random vectors with mean zero and positive definite covariance matrix $\boldsymbol{\Sigma}_{\boldsymbol{e}}$, and $\boldsymbol{e}_t$ is independent of the historical observations $\boldsymbol{y}_{t-1}, \boldsymbol{y}_{t-2}, \cdots$.*

Condition 1 is sufficient and necessary for the strict stationarity of the linear autoregressive process. The Gaussian assumption in Condition 2 is very common in high-dimensional time series literature for technical convenience.

Multiple sequence data may exhibit strong temporal and inter-sequence dependence. To analyze how dependence in the data affects the learning efficiency, we follow Basu and Michailidis (2015) to use the spectral measure of dependence below.

Definition 1. *Define the matrix polynomial $\mathcal{A}(z) = \boldsymbol{I}_N - \boldsymbol{A}_1 z - \cdots - \boldsymbol{A}_P z^P$, where z is any point on the complex plane, and define its extreme eigenvalues as*

$$\mu_{\min}(\mathcal{A}) := \min_{|z|=1} \lambda_{\min}(\mathcal{A}^*(z)\mathcal{A}(z)),$$

$$\mu_{\max}(\mathcal{A}) := \max_{|z|=1} \lambda_{\max}(\mathcal{A}^*(z)\mathcal{A}(z)),$$

where $\mathcal{A}^*(z)$ is the Hermitian transpose of $\mathcal{A}(z)$.

By Condition 1, the extreme eigenvalues are bounded away from zero and infinity, $0 < \mu_{\min}(\mathcal{A}) \le \mu_{\max}(\mathcal{A}) < \infty$. Based on the spectral measure of dependence, we can derive the non-asymptotic statistical convergence rates for the LR and LTR estimators. Note that C denotes a generic positive constant, which is independent of dimension and sample size, and may represent different values. For any positive number a and b, $a \lesssim b$ and $a \gtrsim b$ denote that there exists C such that $a < Cb$ and $a > Cb$, respectively.

Theorem 1. *Suppose that Conditions 1-2 are satisfied, and the sample size $T \gtrsim r_1 r_2 r_3 + N r_1 + N r_2 + P r_3$. With probability at least $1 - \exp[-C(r_1 r_2 r_3 + N r_1 + N r_2 + P r_3)] - \exp(-C\sqrt{T})$,*

$$\|\widehat{\mathcal{W}}_{\text{LTR}} - \mathcal{W}_0\|_{\text{F}} \lesssim \mathcal{M} \sqrt{\frac{r_1 r_2 r_3 + N r_1 + N r_2 + P r_3}{T}},$$

where $\mathcal{M} := [\lambda_{\max}(\boldsymbol{\Sigma}_{\boldsymbol{e}}) \mu_{\max}(\mathcal{A})] / [\lambda_{\min}(\boldsymbol{\Sigma}_{\boldsymbol{e}}) \mu_{\min}^{1/2}(\mathcal{A})]$ is the dependence measure constant.

Theorem 2. *Suppose that Conditions 1-2 are satisfied, $r \ge r_1$ and the sample size $T \gtrsim r(N + NP)$. With probability at least $1 - \exp[-Cr(N + NP))] - \exp(-C\sqrt{T})$,*

$$\|\widehat{\boldsymbol{W}}_{\text{LR}} - \boldsymbol{W}_0\|_{\text{F}} \lesssim \mathcal{M} \sqrt{\frac{r(N + NP)}{T}}.$$

The proofs of Theorems 1 and 2 are provided in the full version, see Wang et al. (2019). The above two theorems present the non-asymptotic convergence upper bounds for

LTR and LR estimators, respectively. Both upper bounds take a general form of $\mathcal{M}\sqrt{d/T}$, where $\mathcal{M}$ captures the effect from dependence across $\boldsymbol{x}_t$, and d denotes the number of parameters in Tucker decomposition or matrix factorization.

Next, we present a minimax lower bound for low-Tucker-rank tensors. The minimax rate is the convergence rate that cannot be improved and has been widely applied in sample complexity analysis. Denote by $\mathbb{T}(r_1, r_2, r_3)$ the set of tensors with ranks at most (r_1, r_2, r_3).

Theorem 3. *Suppose that Condition 2 is satisfied,* $r_1 \leq N/3$, $r_2 \leq N/3$, $r_3 \leq P/3$, *and* $4r_k \leq (r_1 r_2 r_3)/r_k$ *for* $k = 1, 2, 3$. *Then*

$$\inf_{\mathcal{S}} \sup_{\mathcal{W}_0 \in \mathbb{T}(r_1, r_2, r_3)} \|\mathcal{S} - \mathcal{W}_0\|_{\mathrm{F}} \gtrsim \mathcal{M}\sqrt{\frac{Nr_1 + Nr_2 + Pr_3}{T}}$$

with probability at least 0.5, where the infimum is taken over all possible estimator $\mathcal{S}$.

By Theorems 1-3, the upper bound of LTR estimator matches the minimax lower bound up to some constant, and the LTR estimator is optimal and more efficient than the LR estimator. For a training error $\epsilon > 0$, the sample complexity is $T = C(r_1 r_2 r_3 + Nr_1 + Nr_2 + Pr_3)/\epsilon^2$ for the LTR estimator to have $\|\widehat{\mathcal{W}}_{\mathrm{LTR}} - \mathcal{W}_0\|_{\mathrm{F}} \leq \epsilon$; while the sample complexity is $T \gtrsim r(N + NP)/\epsilon^2$ for the LR estimator. Similarly, the OLS estimator can be shown to have the convergence rate of $O(\sqrt{N^2 P/T})$, and its sample complexity is $CN^2 P/\epsilon^2$ to achieve a training error ϵ.

The sample complexity for the linear autoregressive networks with different structures is proportional to the corresponding model complexity, i.e., sample complexity is $O(\mathcal{M}d/\varepsilon^2)$. Compared with the OLS estimator, the LR and LTR estimators benefit from the compact low-dimensional structure and have smaller sample complexity. Among the three linear autoregressive networks, the LTR network has the most compact structure, and hence the smallest sample complexity.

The sample complexity analysis of the autoregressive networks can be extended to the general feed-forward networks for independent $\boldsymbol{x}_t$ by replacing $\mathcal{M}$ with the inverse of signal-to-noise ratio, and explains why the low-rank structure can enhance the learning efficiency and reduce the sample complexity.

Tucker Autoregressive Net

This section introduces a compact autoregressive network by formulating the linear autoregressive network with the low-Tucker-rank structure (5), and it has a compact multi-layer CNN architecture. We call it the Tucker AutoRegressive (TAR) net for simplicity.

Network Architecture

Rather than directly constraining the matrix rank or Tucker ranks of weights in the zero-hidden-layer network, we can modify the network architecture by adding convolutional layers and fully connected layers to exploit low-rank structure. By some algebra, the framework (5) can be rewritten into

$$\boldsymbol{h}_t = \boldsymbol{U}_1 \mathcal{G}_{(1)} \mathrm{vec}(\boldsymbol{U}_2^\top \boldsymbol{X}_t \boldsymbol{U}_3) + \boldsymbol{b},$$

where $\boldsymbol{X}_t = (\boldsymbol{y}_{t-1}, \ldots, \boldsymbol{y}_{t-P})$. A direct translation of the low-Tucker-rank structure leads to a multi-layer convolutional network architecture with two convolutional layers and two fully connected layers; see Figure 2 and Table 1.

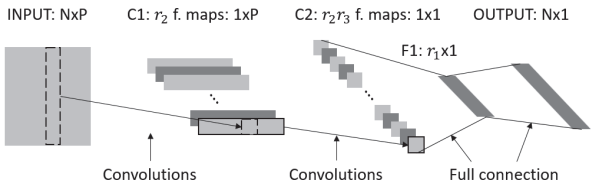

Figure 2: CNN structure of TAR net.

To be specific, each column in $\boldsymbol{U}_2$ is a $N \times 1$ kernel and the first layer outputs r_2 $1 \times P$ feature maps. Similarly, $\boldsymbol{U}_3$ represents the convolution with kernel size $1 \times P$ and r_3 channels. These two convolutional layers work as an encoder to extract the $r_2 r_3$-dimensional representation of the $N \times P$ input $\boldsymbol{X}_t$ for predicting $\boldsymbol{y}_t$. Next, a full connection from $r_2 r_3$ predictor features to r_1 output features with weights $\mathcal{G}_{(1)}$ is followed. Finally, a fully connected layer serves as a decoder to N ouputs with weights $\boldsymbol{U}_1$.

The neural network architectures corresponding to the low-rank estimator $\widehat{\boldsymbol{W}}_{\mathrm{LR}}$ and ordinary least squares estimator without low-dimensional structure $\widehat{\boldsymbol{W}}_{\mathrm{OLS}}$ are the one-hidden-layer MLP with a bottleneck layer of size r and the zero-hidden-layer fully connected network, respectively.

The CNN representation in Figure 2 has a compact architecture with $r_1 r_2 r_3 + Nr_1 + Nr_2 + Pr_3$ parameters, which is the same as that of the Tucker decomposition. Compared with the benchmark models, i.e., the one-hidden-layer MLP (MLP-1) and zero-hidden-layer MLP (MLP-0), the introduced low-Tucker-rank structure increases the depth of the network while reduces the total number of weights. When Tucker ranks are small, the total number of parameters in our network is much smaller than those of the benchmark networks, which are $r(N + NP)$ and $N^2 P$, respectively.

To capture the complicated and non-linear functional mapping between the prior inputs and future responses, non-linear activation functions, such as rectified linear unit (ReLU) or sigmoid function, can be added to each layer in the compact autoregressive network. Hence, the additional depth from transforming a low-Tucker-rank single layer to a multi-layer convolutional structure enables the network to approximate the target function better. The linear network without activation in the previous section can be called the linear TAR net (LTAR).

Separable Convolutional Kernels

Separable convolutions have been extensively studied to replace or approximate large convolutional kernels by a series of smaller kernels. For example, this idea was explored in multiple iterations of the Inception blocks (Szegedy et al.

Symbol	Feedforward	Layer	Content and explanation	Dimensions	No. of parameters
INPUT	$\boldsymbol{X}_t$	-	design matrix	$N \times P$	-
C1	$\boldsymbol{Z}_1 := \boldsymbol{U}_2^\top \boldsymbol{X}_t$	$N \times 1$ convolutions	r_2 feature maps	$1 \times P$	Nr_2
C2	$\boldsymbol{Z}_2 := \boldsymbol{U}_3^\top \boldsymbol{Z}_1^\top$	$1 \times P$ convolutions	$r_2 r_3$ feature maps	1×1	Pr_3
F1	$\boldsymbol{Z}_3 := \mathcal{G}_{(1)}\text{vec}(\boldsymbol{Z}_2^\top)$	full connection	response factor loadings	$r_1 \times 1$	$r_1 r_2 r_3$
OUTPUT	$\boldsymbol{Z}_4 := \boldsymbol{h}_t = \boldsymbol{U}_1 \boldsymbol{Z}_3$	full connection	output prediction	$N \times 1$	Nr_1

Table 1: Specification of CNN structure in TAR net.

2015; 2016; 2017) to decompose a convolutional layer with a 7×7 kernel into that with 1×7 and 7×1 kernels.

Tensor decomposition is an effective method to obtain separable kernels. In our TAR net, these two convolutional layers extract the information from inputs along the column-wise direction and row-wise direction separately. Compared with the low-rank matrix structure, the additional decomposition in the Tucker decomposition along the second and third modes segregates the full-sized convolutional kernel into $r_2 r_3$ pairs of separable kernels.

Two-Lane Network

If no activation function is added, the first two row-wise and column-wise convolutional layers are exchangeable. However, exchanging these two layers with nonlinear activation functions can result in different nonlinear approximation and physical interpretation.

For the general case where we have no clear preference on the order of these two layers, we consider a two-lane network variant, called TAR-2 network, by introducing both structures into our model in parallel followed by an average pooling to enhance the flexibility; see Figure 3 in Wang et al. (2019) for a graphical illustration.

Implementation

Details We implement our framework on PyTorch, and the Mean Squared Error (MSE) is the target loss function. The gradient descent method is employed for the optimization with learning rate and momentum being 0.01 and 0.9, respectively. If the loss function drops by less than 10^{-8}, the procedure is then deemed to have reached convergence.

Hyperparameter tuning In the TAR net, the sequential dependence range P and the Tucker ranks r_i, are prespecified hyperparameters. Since cross-validation cannot be applied to sequence modeling, we suggest tuning hyperparameters by grid search and rolling forecasting performance.

Experiments

This section first performs analysis on two synthetic datasets to verify the sample complexity established in Theorems 1-3 and to demonstrate the capability of TAR nets in nonlinear functional approximation. Three real datasets are then analyzed by the TAR-2 and TAR nets, together with their linear counterparts. For the sake of comparison, some benchmark networks, including MLP-0, MLP-1, Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM), are also applied to the dataset.

Numerical Analysis for Sample Complexity

Settings In TAR net or the low-Tucker-rank framework (5), the hyperparameters, r_1, r_2 and r_3, are of significantly smaller magnitude than N or P, and are equally set to 2 or 3. As sample complexity is of prime interest rather than the range of sequential dependence, we let P equal to $3, 5$ or 8. For each combination of (r_1, r_2, r_3, P), we consider $N = 9, 25$ and 36, and the sample size T is chosen such that $\sqrt{N/T} = (0.15, 0.25, 0.35, 0.45)$.

Data generation We first generate a core tensor $\mathcal{G} \in \mathbb{R}^{r_1 \times r_2 \times r_3}$ with entries being independent standard normal random variables, and then rescale it such that the largest singular value of $\mathcal{G}_{(1)}$ is 0.9. For each $1 \le i \le 3$, the leading r_i singular vectors of random standard Gaussian matrices are used to form $\boldsymbol{U}_i$. The weight tensor $\mathcal{W}_0$ can thereby be reconstructed, and it is further rescaled to satisfy Condition 1. We generate 200 sequences with identical $\mathcal{W}_0$. The first 500 simulated data points at each sequence are discarded to alleviate the influence of the initial values. We apply the MLP-0, MLP-1 and LTAR to the synthetic dataset. The averaged estimation errors for the corresponding OLS, LR, and LTR estimators are presented in Figure 3.

Results The x-axis in Figure 3 represents the ratio of $\sqrt{N/T}$, and the y-axis represents the averaged estimation error in Frobenius norm. Along each line, as N is set to be fixed, we obtain different points by readjusting the sample size T. Roughly speaking, regardless of the models and parameter settings, estimation error increases with varying rates as the sample size decreases. The rates for OLS rapidly become explosive, followed by LR, whereas LTR remains approximately linear, which is consistent with our findings at Theorems 1 and 2.

Further observation reveals that the increase in P predominantly accelerates the rates for OLS and LR, but appears to have an insignificant influence on the estimation error from LTR.

For the case with $P = 8$, instability of the estimation error manifests itself in LR under insufficient sample size, say when $\sqrt{N/T}$ is as large as 0.35. This further provides the rationale for dimension reduction along sequential order. When $\sqrt{N/T} = 0.45$, the solution is not unique for both OLS and LR, and consequently, these points are not shown in the figure.

Numerical Analysis for Nonlinear Approximation

Settings The target of this experiment is to compare the expressiveness of LTAR, TAR and TAR-2 nets. The conjecture is that, regardless of the data generating process, TAR-2 and TAR nets under the same hyperparameter settings as the LTAR net would have an elevated ability to capture nonlinear features. We set $(r_1, r_2, r_3, N, P) = (2, 2, 2, 25, 3)$, and have also tried several other combinations. Similar findings can be observed, and the results are hence omitted here.

Data generation Two data generating processes are considered to create sequences with either strictly linear or highly nonlinear features in the embedding feature space. We refer to them as L-DGP and NL-DGP, respectively. L-DGP is achieved by randomly assigning weights to LTAR layers and producing a recursive sequence with a given initial input matrix. NL-DGP is attained through imposing a nonlinear functional transformation to the low-rank hidden layer of an MLP. In detail, we first transformed a $N \times P$ matrix to a $r_1 \times r_2$ low-rank encoder. Then, we applied a nonlinear mapping $f(\cdot) = \cos(1/\|\cdot\|_{\mathrm{F}})$ to the encoder, before going through a fully connected layer to retrieve an output of size $N \times 1$.

Implementation & Evaluation In this experiment, we use L-DGP and NL-DGP to separately generate 200 data sequences, which are fitted by TAR-2, TAR and LTAR nets. The sequence lengths are chosen to be either 101 or 501. For each sequence, the last data point is retained as a single test point, whereas the rest are used in model training. We adopt three evaluation metrics, namely, the averaged L_2 norm between prediction and true value, the standard Root-Mean-Square Error (RMSE), and Mean Absolute Error (MAE). The results are given in Table 2.

Results When the data generating process is linear (L-DGP), the LTAR net reasonably excels in comparison to the other two, obtaining the smallest L_2-norm, RMSE and MAP. TAR-2 yields poorer results for a small sample size of 100 due to possible overparametrization. However, its elevated expressiveness leads it to outperform TAR when $T = 500$.

For the nonlinear data generating process (NL-DGP), as we expect, the TAR-2 and TAR nets with nonlinear structure outperform the LTAR net. In the meanwhile, as the exchangeability of latent features holds, the TAR-2 net seems to suffer from model redundancy and thereby performs worse than the TAR net.

Real Datasets

Dataset We use the three publicly available datasets.

1. USME dataset: It contains 40 US macroeconomic variables provided in Koop (2013), including consumption, production indices, stock market indicators and the interest rates. The data series are taken quarterly from 1959 to 2007 with a total of 194 observed time points.
2. GHG dataset: We retrieve a partial greenhouse gas (GHG) concentration dataset (Lucas et al. 2015) from UCI repository containing the time series of GHG tracers. The observations are gathered from 45 sensors, averaged to be spaced 12 hours apart with 327 time points.

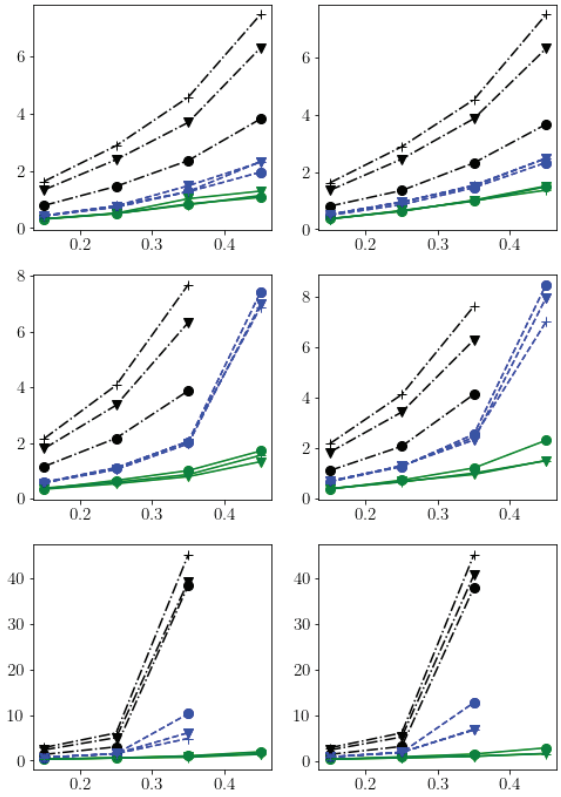

Figure 3: Experiment on sample complexity. Results are shown for OLS (·—·), LR (– –) and LTR (——) estimators. Three different values of N are presented by different markers: $N = 9$ (•), $N = 25$ (▼) and $N = 36$ (+). We set $(r_1, r_2, r_3) = (2, 2, 2)$ for subplots in the left column and $(3, 3, 3)$ for subplots in the right column. And upper, middle and lower panels refer to cases with $P = 3, 5$ and 9.

3. Traffic dataset: The data record the hourly road occupancy rate (between 0 and 1) in the San Francisco Bay freeway (Lai et al. 2018) obtained by 30 different sensors in 2015.

In the preprocessing step, the series were transformed to be stationary before further standardization; details see the full version (Wang et al. 2019).

Models for comparison For the sake of comparison, besides the proposed models, TAR-2, TAR and LTAR, we also consider four other commonly used networks in the literature with well-tuned hyperparameters. The first two are the previously mentioned MLP-0 and MLP-1. The remaining two are RNN and LSTM, which are two traditional sequence modeling frameworks. RNN implies an autoregressive moving average framework and can transmit extra useful information through the hidden layers. It is hence expected to outperform an autoregressive network. LSTM is advantageous in many sequence learning tasks, but may be more susceptible to small sample size. As a result, RNN and LSTM with the optimal tuning hyperparameters serve as our benchmarks.

DGP	T	Network	L_2-norm	RMSE	MAP
L	100	TAR-2	5.5060	1.1238	0.8865
		TAR	5.4289	1.0998	0.8702
		LTAR	**5.1378**	**1.0388**	**0.8265**
	500	TAR-2	5.1836	1.0493	0.8369
		TAR	5.2241	1.0585	0.8436
		LTAR	**4.9338**	**0.9972**	**0.7936**
NL	100	TAR-2	5.2731	**1.0703**	0.8579
		TAR	**5.2710**	1.0712	**0.8510**
		LTAR	5.3161	1.0738	0.8573
	500	TAR-2	5.0084	1.0111	0.8062
		TAR	**5.0036**	**1.0110**	**0.8060**
		LTAR	5.0144	1.0126	0.8087

Table 2: Performance of different networks on fitting L-DGP and NL-DGP datasets.

Implementation We tune the parameters (P, r_1, r_2, r_3) for our proposed models. To ensure a fair comparison, the size of the hidden layer in MLP-1 is set to be P, and we consider only one hidden layer for RNN and LSTM. The number of neurons in the hidden layer is treated as a tunable hyperparameter. The bias terms are added back to the TAR-2, TAR and LTAR nets for expansion of the model space.

For each network, one-step-ahead forecasting is carried out recursively. In other words, the trained network predicts one future step, and immediately includes the new observation in the training set for the prediction of the next step. The averaged L_2-norm, RMSE and MAP are used as the evaluation criteria.

For the USME dataset, the parameters are chosen as $(P, r_1, r_2, r_3) = (4, 4, 3, 2)$. The first 104 time points of each series are used as the first training samples with an effective sample size of 100, whereas the rolling forecast procedure is applied to the rest 90 test samples. For GHG dataset, the parameters are set as $(P, r_1, r_2, r_3) = (20, 2, 4, 5)$ with an effective training sample size of 100 and testing sample size of 207. For the Traffic dataset, the parameters are set as $(P, r_1, r_2, r_3) = (30, 2, 6, 8)$ and we choose a larger training size of 200 with 70 testing samples.

T-test for significance We adopt a paired two-sample t-test for the significant difference of L_2-norm between TAR-2 and RNN/LSTM, separately. The hypotheses are H_0: μ_{L_2}(TAR-2) $=$ μ_{L_2}(RNN/LSTM) against μ_{L_2}(TAR-2) $>$ μ_{L_2}(RNN/LSTM). Note that each forecast error is a vector of $\mathbb{R}^N$, and there are n steps of prediction. Consider the L_2-norm for each forecast error, and it then leads to a sequence of n scalars, which can be assumed to be independent; see Kuester, Mittnik, and Paolella (2006).

For the USME data, the p-value of the test between TAR-2 and LSTM is 0.031, and between TAR-2 and RNN is 0.203. For the GHG data, the tests for comparing TAR-2 to both RNN and LSTM are significant with p-values smaller than 0.01. For the Traffic data, the test between TAR-2 and LSTM has a p-value close to 0. While for the comparison of TAR-2 and RNN, the p-value is 0.052.

Dataset	Network	L_2-norm	RMSE	MAE
USME	MLP-0	11.126	1.8867	1.3804
	MLP-1	7.8444	1.3462	1.0183
	RNN	5.5751	0.9217	0.7064
	LSTM	5.8274	0.9816	0.7370
	LTAR	5.5257	0.9292	0.6857
	TAR	5.4675	0.9104	0.6828
	TAR-2	**5.4287**	**0.8958**	**0.6758**
GHG	MLP-0	0.8025	0.1314	0.0927
	MLP-1	0.7674	0.1387	0.0850
	RNN	0.7420	0.1255	0.0807
	LSTM	0.7608	0.1277	0.0852
	LTAR	0.8072	0.1546	0.0922
	TAR	0.7118	0.1208	0.0758
	TAR-2	**0.7115**	**0.1206**	**0.0758**
Traffic	MLP-0	0.8482	0.1224	0.0889
	MLP-1	0.8451	0.1391	0.0869
	RNN	0.8235	0.1249	0.0802
	LSTM	0.8334	0.1227	0.0821
	LTAR	1.0125	0.1738	0.1057
	TAR	0.7974	**0.1181**	0.0753
	TAR-2	**0.7936**	0.1182	**0.0753**

Table 3: Performance comparison on three real datasets.

Results From Table 3, the proposed TAR-2 and TAR nets rank top two in terms of one-step-ahead rolling forecast performance, exceeding the fine-tuned RNN model with the size of the hidden layer equal to one. The two-lane network TAR-2 outperforms the one-lane network TAR emphasizing its ability to capture non-exchangeable latent features. According to our experiments, the performance of both RNN and LSTM deteriorates as the dimension of the hidden layer increases, which indicates that overfitting is a serious issue for these two predominate sequence modeling techniques.

Conclusion and Discussion

This paper rearranges the weights of an autoregressive network into a tensor, and then makes use of the Tucker decomposition to introduce a low-dimensional structure. A compact autoregressive network is hence proposed to handle the sequences with long-range dependence. Its sample complexity is also studied theoretically. The proposed network can achieve better prediction performance on a macroeconomic dataset than some state-of-the-art methods including RNN and LSTM.

For future research, this work can be improved in three directions. First, our sample complexity analysis is limited to linear models, and it is desirable to extend the analysis to networks with nonlinear activation functions. Secondly, the dilated convolution in WaveNet (Van den Oord et al. 2016a), can reduce the convolutional kernel size, and hence can efficiently access the long-range historical inputs. This structure can be incorporated into our framework to further compress the network. Finally, a deeper autoregressive network can be constructed to enhance the expressiveness of nonlinearity.

References

Basu, S., and Michailidis, G. 2015. Regularized estimation in sparse high-dimensional time series models. *The Annals of Statistics* 43(4):1535–1567.

Cho, K.; Van Merriënboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; and Bengio, Y. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *Empirical Methods in Natural Language Processing*.

Denil, M.; Shakibi, B.; Dinh, L.; De Freitas, N.; et al. 2013. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems*, 2148–2156.

Du, S. S.; Wang, Y.; Zhai, X.; Balakrishnan, S.; Salakhutdinov, R. R.; and Singh, A. 2018. How many samples are needed to estimate a convolutional neural network? In *Advances in Neural Information Processing Systems*, 373–383.

Fontaine, V.; Ris, C.; and Boite, J.-M. 1997. Nonlinear discriminant analysis for improved speech recognition. In *Fifth European Conference on Speech Communication and Technology*.

Goodfellow, I.; Bengio, Y.; and Courville, A. 2016. *Deep learning*. MIT press.

Grézl, F.; Karafiát, M.; Kontár, S.; and Cernocky, J. 2007. Probabilistic and bottle-neck features for lvcsr of meetings. In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07*, volume 4, IV–757. IEEE.

Hinton, G. E., and Salakhutdinov, R. R. 2006. Reducing the dimensionality of data with neural networks. *Science* 313(5786):504–507.

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural Computation* 9(8):1735–1780.

Jaderberg, M.; Vedaldi, A.; and Zisserman, A. 2014. Speeding up convolutional neural networks with low rank expansions. In *Proceedings of the British Machine Vision Conference*. BMVA Press.

Kim, Y.-D.; Park, E.; Yoo, S.; Choi, T.; Yang, L.; and Shin, D. 2016. Compression of deep convolutional neural networks for fast and low power mobile applications. In *International Conference on Learning Representations*.

Kolda, T. G., and Bader, B. W. 2009. Tensor decompositions and applications. *SIAM Review* 51(3):455–500.

Koop, G. M. 2013. Forecasting with medium and large bayesian vars. *Journal of Applied Econometrics* 28(2):177–203.

Kuester, K.; Mittnik, S.; and Paolella, M. S. 2006. Value-at-risk prediction: A comparison of alternative strategies. *Journal of Financial Econometrics* 4:53–89.

Lai, G.; Chang, W.-C.; Yang, Y.; and Liu, H. 2018. Modeling long-and short-term temporal patterns with deep neural networks. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, 95–104. ACM.

Lebedev, V.; Ganin, Y.; Rakhuba, M.; Oseledets, I.; and Lempitsky, V. 2014. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553*.

Lucas, D.; Yver Kwok, C.; Cameron-Smith, P.; Graven, H.; Bergmann, D.; Guilderson, T.; Weiss, R.; and Keeling, R. 2015. Designing optimal greenhouse gas observing networks that consider performance and cost. *Geoscientific Instrumentation, Methods and Data Systems* 4(1):121–137.

Novikov, A.; Podoprikhin, D.; Osokin, A.; and Vetrov, D. P. 2015. Tensorizing neural networks. In *Advances in Neural Information Processing Systems*, 442–450.

Pan, Y.; Xu, J.; Wang, M.; Ye, J.; Wang, F.; Bai, K.; and Xu, Z. 2019. Compressing recurrent neural networks with tensor ring for action recognition. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 4683–4690.

Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; and Rabinovich, A. 2015. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1–9.

Szegedy, C.; Vanhoucke, V.; Ioffe, S.; Shlens, J.; and Wojna, Z. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2818–2826.

Szegedy, C.; Ioffe, S.; Vanhoucke, V.; and Alemi, A. A. 2017. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-First AAAI Conference on Artificial Intelligence*.

Tjandra, A.; Sakti, S.; and Nakamura, S. 2018. Tensor decomposition for compressing recurrent neural network. In *2018 International Joint Conference on Neural Networks*, 1–8.

Tucker, L. R. 1966. Some mathematical notes on three-mode factor analysis. *Psychometrika* 31(3):279–311.

Van den Oord, A.; Dieleman, S.; Zen, H.; Simonyan, K.; Vinyals, O.; Graves, A.; Kalchbrenner, N.; Senior, A.; and Kavukcuoglu, K. 2016a. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*.

Van den Oord, A.; Kalchbrenner, N.; Espeholt, L.; Vinyals, O.; Graves, A.; et al. 2016b. Conditional image generation with pixelcnn decoders. In *Advances in Neural Information Processing Systems*, 4790–4798.

Wang, D.; Huang, F.; Zhao, J.; Li, G.; and Tian, G. 2019. Compact autoregressive network. *arXiv preprint arXiv:1909.03830*.

Ye, J.; Wang, L.; Li, G.; Chen, D.; Zhe, S.; Chu, X.; and Xu, Z. 2018. Learning compact recurrent neural networks with block-term tensor decomposition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 9378–9387.