Adaptive Detrending to Accelerate Convolutional Gated Recurrent Unit Training for Contextual Video Recognition

Minju Jung[a,c], Haanvid Lee[b], Jun Tani[c,*]

[a] *School of Electrical Engineering, Korea Advanced Institute of Science and Technology, Daejeon, Korea*
[b] *School of Computing, Korea Advanced Institute of Science and Technology, Daejeon, Korea*
[c] *Cognitive Neurorobotics Research Unit, Okinawa Institute of Science and Technology Graduate University, Okinawa, Japan*

Abstract

Video image recognition has been extensively studied with rapid progress recently. However, most methods focus on short-term rather than long-term (contextual) video recognition. Convolutional recurrent neural networks (ConvRNNs) provide robust spatio-temporal information processing capabilities for contextual video recognition, but require extensive computation that slows down training. Inspired by normalization and detrending methods, in this paper we propose "adaptive detrending" (AD) for temporal normalization in order to accelerate the training of ConvRNNs, especially of convolutional gated recurrent unit (ConvGRU). For each neuron in a recurrent neural network (RNN), AD identifies the trending change within a sequence and subtracts it, removing the internal covariate shift. In experiments testing for contextual video recognition with ConvGRU, results show that (1) ConvGRU clearly outperforms feed-forward neural networks, (2) AD consistently and significantly accelerates training and improves generalization, (3) performance is further improved when AD is coupled with other normalization methods, and most importantly, (4) the more long-term contextual information is required, the more AD outperforms existing methods.

*Corresponding author
Email address: tani1216jp@gmail.com (Jun Tani)

Keywords: Detrending, normalization, internal covariate shift, convolutional neural networks (CNNs), recurrent neural networks (RNNs), convolutional recurrent neural networks (ConvRNNs)

1. Introduction

Convolutional neural networks (CNNs) [1] show remarkable performance on the ImageNet challenge dataset, consisting of 1000 classes and 1.2 million training images [2]. Encouraged by this success, several approaches exploit the spatial processing capability of CNNs in video recognition tasks [3, 4]. Two-stream CNNs [3] and convolutional 3D (C3D) networks [4] are the most commonly used networks. Two-stream CNNs combine classification abilities of spatial- and temporal-stream networks, being composed of a spatial-stream network that processes individual RGB frames and a temporal-stream network that processes stacked optical flow over several frames. C3D networks extend 2D convolution to 3D convolution by adding time as a third dimension, processing stacked consecutive RGB frames. However, both networks employ a stacking strategy that utilizes only a limited number of temporal correlations between stacked frames in order to recognize videos. Once the temporal window advances to the next position, information from the previous stack is completely dropped. This creates a problem of contextual recognition that requires the extraction of long-range temporal correlations [5].

In this paper, we attempt to overcome this limitation using recently introduced convolutional recurrent neural networks (ConvRNNs) that replace the weight multiplication of RNNs with convolution in order to exploit spatial and temporal information processing capabilities of CNNs and recurrent neural networks (RNNs), respectively [6, 7, 8]. By extracting spatio-temporal features hierarchically, ConvRNNs handle complex problems in the space-time domain, such as precipitation nowcasting [6], video recognition [7], and video prediction [8]. Also, problems restricted to the spatial domain can be handled by ConvRNNs in an iterative manner [9]. For example, in instance segmentation,

ConvRNNs sequentially segment one instance of an image at a time [9]. However, training ConvRNNs is painfully slower than training feed-forward CNNs, which receive a single frame or stacked multiple frames for video recognition, because recurrent connections require additional computation. Moreover, it is hard to parallelize computation of ConvRNNs due to the sequential nature of RNNs, which require computations from previous time steps in advance for computing the current time step. Thus, finding a way to achieve faster learning convergence has been a barrier to practical development of ConvRNNs.

Loffe and Szegedy [10] argue that internal covariate shift is responsible for the increased training time in feed-forward neural networks, including multi-layer perceptrons (MLPs) and CNNs, and they suggest batch normalization (BN) to normalize the input distribution of a neuron for each mini-batch, as a way to reduce training time. BN successfully removes internal covariate shift, thereby significantly accelerating training with improved generalization, and this technique has become standard for training feed-forward neural networks. Some studies use BN with RNNs because unrolled RNNs over time can be seen as deep neural networks in terms of time as well as depth [11, 12]. However, BN is incompatible with RNNs, regardless of computing global statistics along the time domain [11] or local statistics at each time step [12]. Use of global statistics ignores statistics at each time step, but uses of local statistics does not accommodate training sequences of variable lengths. As an alternative, layer normalization (LN) [13] eliminates dependencies between mini-batch samples that obviate the use of BN with RNNs. LN computes statistics over all neurons in each layer and accelerates training of RNNs and MLPs, but not CNNs. Neither BN nor LN is generally applied to ConvRNNs.

The current paper focuses on the time domain in order to accelerate training of ConvRNNs. Much of time series analysis and many forecasting methods can be applied only to stationary time series. Detrending transforms non-stationary time series to stationary series by identifying the change as a trend and removing it. This method is straightforward, and is illustrated in the context of the

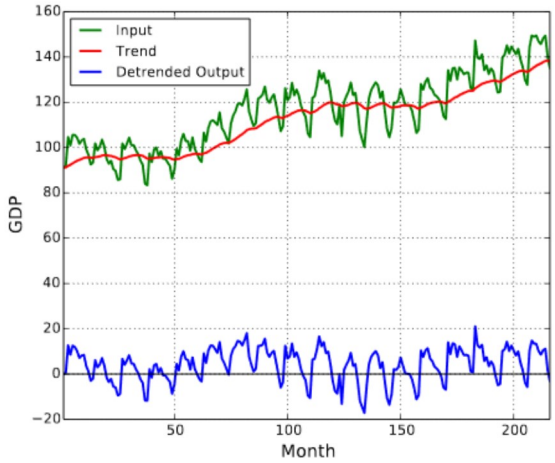

Figure 1: Example of conventional detrending with Brazilian GDP. The detrended output is obtained by subtracting the trend from the original input. In this example, we use an exponential moving average (EMA) with a fixed decay factor of 0.95 to define the trend.

Brazilian gross domestic product[1] in Fig. 1. The current research applies this method to normalize sequences of neurons in RNNs. Our key insight here is that the hidden state of a gated recurrent unit (GRU) [14] can be considered as a trend that can be approximated by the form of an exponential moving average with an adaptively changing decay factor. Based on this insight, we propose a novel temporal normalization method, "adaptive detrending" (AD), for use with GRU and convolutional gated recurrent unit (ConvGRU), which is a variant of ConvRNNs extended from GRU. The implications of AD are fourfold:

- AD is easy to implement, reducing computational cost and consuming less memory than competing methods.
- AD eliminates temporal internal covariate shift.
- AD controls the degree of detrending (or normalization) through decay factor adaptability.

[1] http://www2.stat.duke.edu/~mw/data-sets/ts_data/brazil_econ

- AD is fully compatible with existing normalization methods.

2. Background

2.1. Batch Normalization

Internal covariate shift slows training of deep neural networks, because the distribution of layer inputs changes continuously as lower layer parameters are updated. Batch normalization (BN) [10] has recently been proposed to reduce internal covariate shift by normalizing network activation as follows:

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x_i \tag{1}$$

$$\sigma^2 = \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu)^2 \tag{2}$$

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} \tag{3}$$

$$y_i = \gamma \hat{x}_i + \beta \tag{4}$$

where x is the activations of a neuron in a mini-batch of size m, μ and σ^2 are the mean and variance of a mini-batch, respectively, $\hat{x}$ is normalized input, ϵ is an infinitesimal constant for numerical stability, and y is an affine transformation of normalized inputs $\hat{x}$. During training, the input distribution to a layer is transformed to a fixed distribution with a zero mean and unit variance, regardless of the change in parameters of lower layers. Additionally, an affine transformation with two learnable parameters γ and β follows normalization in order to recover the original activation when required. BN accelerates training and improves generalization of CNNs on ImageNet classification tasks.

Due to its success in feed-forward neural networks, BN has been applied to RNNs to speed training and improve generalization [11, 12]. In [11], BN is applied only to vertical (input-to-hidden) and not to horizontal (hidden-to-hidden) connections because the repeated rescaling of horizontal connections

induces vanishing and exploding gradient problems. Also, the mean and variance for BN are computed by averaging along not only the mini-batch axis but also the time axis, which is called "sequence-wise normalization." On the other hand, Cooijmans et al. [12] develop "step-wise normalization" and show that (1) applying BN to horizontal as well as vertical connections is possible by properly initializing γ of an affine transformation and beneficial for reducing temporal internal covarite shift, and (2) using statistics for each time step separately preserves initial transient phase information. However, with this method, estimation of statistics at each time step degrades along the time axis due to variation in length of training and test sequences. During training, mini-batch configuration involves the use of zero, or last frame padding for shorter sequences. Furthermore, statistics for each time step are estimated only up to the length of the longest training sequence T_{max}. After training, accurate statistics for test sequences longer than the longest training sequence T_{max} cannot be generated. Due to these factors, performance suffers.

2.2. Layer Normalization

Ba et al. [13] introduce "layer normalization" (LN) to overcome the limitations of BN when applied to RNNs. LN has the same form as that of Cooijmans et al.'s [12] step-wise normalization, with the difference that LN normalizes over the spatial axis rather than by mini-batch. The assumption underlying LN is that changes in output from one layer correlate highly with changes in summed inputs of the next layer. Hence, LN estimates statistics for data from a single training session using all activations in each layer. By estimating statistics over layers instead of mini-batches, LN properly estimates statistics at each time step, regardless of mini-batch sequence length variability. In experiments with RNNs, LN achieves faster convergence and better generalization than baseline and other normalization methods, especially for long sequences and small mini-batches.

However, LN does not perform well with CNNs. The authors report that LN is better than the baseline without normalization, but not better than BN.

They hypothesize that neurons in a layer have different statistics due to the spatial topology of feature maps, so that the central assumption of LN cannot be supported for CNNs. We agree that normalizing all neurons in a layer with the same statistics is not the best method for normalizing CNNs. However, because BN works successfully for CNNs by estimating statistics of each feature map, LN's shortcomings with CNNs might reflect different statistics between feature maps, and not within a feature map.

3. Model

3.1. Gated Recurrent Unit

Standard recurrent neural networks (RNNs) have greater utility than feed-forward networks because they add a recurrent connection to handle sequential data. RNNs consist of three layers: an input layer $\mathbf{x}$, a hidden layer $\mathbf{h}$, and an output layer $\mathbf{y}$. RNNs are able to handle sequential data because the hidden layer receives both current input from the input layer as well as information about its own previous state through a recurrent connection as follows:

$$\mathbf{h}_t = g(\mathbf{W}_h \mathbf{x}_t + \mathbf{U}_h \mathbf{h}_{t-1} + \mathbf{b}_h) \tag{5}$$

$$\mathbf{y}_t = f(\mathbf{W}_y \mathbf{h}_t + \mathbf{b}_y) \tag{6}$$

where $g(\cdot)$ and $f(\cdot)$ are element-wise non-linear activation functions for the hidden and output layers, respectively, and $\mathbf{W}$, $\mathbf{U}$, and $\mathbf{b}$ represent the learnable parameters of RNNs: forward connection weights, recurrent connection weights, and biases, respectively.

However, standard RNNs do not capture long-term dependencies well because of vanishing and exploding gradient problems [15, 16]. The gated recurrent unit (GRU) was proposed by Cho et al. [14] to overcome the vanishing gradient problem. It employs the same gating mechanism as long short-term memory (LSTM) [17], but employs a simpler architecture by eliminating the output gate and modifying some other parts of LSTM. Specifically, GRU has

two gating units, called a reset gate $\mathbf{r}$ and an update gate $\mathbf{z}$. The hidden state $\mathbf{h}_t$ at each time step t is calculated using a leaky integrator with an adaptive time constant determined by the update gate $\mathbf{z}$. In other words, the hidden state $\mathbf{h}_t$ is a linear interpolation between the previous hidden state $\mathbf{h}_{t-1}$ and the candidate hidden state $\tilde{\mathbf{h}}_t$ as weighted by the update gate $\mathbf{z}$, and is defined as follows:

$$\mathbf{h}_t = \mathbf{z}_t \boxed{} \tilde{\mathbf{h}}_t + (1 - \mathbf{z}_t) \boxed{} \mathbf{h}_{t-1} \tag{7}$$

$$\mathbf{z}_t = \sigma(\mathbf{W}_z \mathbf{x}_t + \mathbf{U}_z \mathbf{h}_{t-1} + \mathbf{b}_z) \tag{8}$$

where $\sigma(\cdot)$ is a sigmoid function and $\boxed{}$ is an element-wise multiplication.

The candidate hidden state $\tilde{\mathbf{h}}_t$ at each time step t is calculated similarly to that of the hidden layer in standard RNNs (5). However, unlike standard RNNs, the reset gate $\mathbf{r}$ determines how much the previous hidden state $\mathbf{h}_{t-1}$ affects the candidate hidden state $\tilde{\mathbf{h}}_t$ as follows:

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_h \mathbf{x}_t + \mathbf{r}_t \boxed{} \mathbf{U}_h \mathbf{h}_{t-1} + \mathbf{b}_h) \tag{9}$$

$$\mathbf{r}_t = \sigma(\mathbf{W}_r \mathbf{x}_t + \mathbf{U}_r \mathbf{h}_{t-1} + \mathbf{b}_h) \tag{10}$$

3.2. Gated Recurrent Unit Normalization in the Spatial Domain

Following Ba et al. [13], in this paper we apply recurrent batch normalization (recurrent BN) [12] and layer normalization (LN) [13] to GRU. We refer to recurrent BN and LN as "spatial" normalization methods to differentiate the present approach from normalization in the time domain reviewed above. The following equations represent GRU normalization in the spatial domain:

$$\mathbf{r}_t = \sigma(\mathrm{N}_{\gamma,\beta}(\mathbf{W}_r \mathbf{x}_t) + \mathrm{N}_{\gamma}(\mathbf{U}_r \mathbf{h}_{t-1})) \tag{11}$$

$$\mathbf{z}_t = \sigma(\mathrm{N}_{\gamma,\beta}(\mathbf{W}_z \mathbf{x}_t) + \mathrm{N}_{\gamma}(\mathbf{U}_z \mathbf{h}_{t-1})) \tag{12}$$

$$\tilde{\mathbf{h}}_t = \tanh(\mathrm{N}_{\gamma,\beta}(\mathbf{W}_h \mathbf{x}_t) + \mathbf{r}_t \boxed{} \mathrm{N}_{\gamma}(\mathbf{U}_h \mathbf{h}_{t-1})) \tag{13}$$