REDUCED-GATE CONVOLUTIONAL LSTM DESIGN USING PREDICTIVE CODING FOR NEXT-FRAME VIDEO PREDICTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Spatiotemporal sequence prediction is an important problem in deep learning. We study next-frame video prediction using a deep-learning-based predictive coding framework that uses convolutional, long short-term memory (convLSTM) modules. We introduce a novel reduced-gate convolutional LSTM architecture. Our reduced-gate model achieves better next-frame prediction accuracy than the original convolutional LSTM while using a smaller parameter budget, thereby reducing training time. We tested our reduced gate modules within a predictive coding architecture on the moving MNIST and KITTI datasets. We found that our reduced-gate model has a significant reduction of approximately 40 percent of the total number of training parameters and training time in comparison with the standard LSTM model which makes it attractive for hardware implementation especially on small devices.

1 INTRODUCTION

The brain in part acquires representations by using learning mechanisms that are triggered by prediction errors found in sensory input (Friston, 2005; Bastos et al., 2012). An early implementation using this approach is used by Rao & Ballard (1999) to model non-classical receptive field properties in the neocortex. The hypothesized brain mechanisms underlying this "predictive coding" are based on the concept of bi-directional interactions between the higher and lower-level areas of the visual cortex. The higher-level areas send predictions about the incoming sensory input to the lower-level areas. The lower-level areas compare the predictions with ground truth sensory input and calculate the prediction errors. These are in turn forwarded to the higher-level areas to update their predictive representations in light of the new input information.

Lotter et al. (2017) introduced a predictive coding architecture, PredNet, for next-frame video prediction. The architecture used a hierarchy of LSTMs. Being a deep network, it could readily be implemented and studied using off-the-shelf deep learning frameworks (e.g. Keras library (2018), PyTorch Paszke et al. (2017), TensorFlow Abadi et al. (2015)). This was in contrast to earlier models with a purely mathematical formulation (Friston, 2005). The LSTM modules in PredNet were based on the convolutional LSTMs (convLSTM) of Shi et al. (2015). However, PredNet used an LSTM architecture (Greff et al., 2017) without peephole connections, whereas the convLSTMs of Shi et al. (2015) included peephole connections. ConvLSTMs differ from regular LSTMs because they process images instead of feature vectors. Since convLSTMs process images, Shi et al. (2015) replaced the matrix multiplication operation with convolution.

In the present paper, our work uses the PredNet architecture (Lotter et al., 2017) but is novel in the LSTM-like modules used to form the architecture. Unlike the modules in PredNet, we use peephole connections. More importantly, we make two novel changes to the Shi et al. (2015) convLSTM design. First, Shi et al. (2015) implemented peephole connections by using the elementwise (Hadamard) multiply (Horn, 1990). Although this technique is used in conventional LSTMs, convolutional LSTMs process images. In a convLSTM, the Hadamard operation will multiply corresponding pixels between two images to create a new image. This is suitable for feature vectors, but for images it creates an excessive number of trainable parameters (N^2 for an $N \times N$ image), and we have found that performance can be improved by replacing the Hadamard operation with con-

volutional operations. Our second modification is to use a novel, reduced-gate convolutional LSTM (rgcLSTM). Specifically, we use one trainable gate (the forget gate) and couple it by equality to the traditional input and output gates. This results in fewer trainable parameters but with equal or better performance accuracy, according to our experiments, than the original PredNet. Our rgcLSTM design is shown in Figure 1. The present paper motivates the design of our rgcLSTM, specifically, the convolution-based peephole connections and the single forget-gate coupled to the input and output gates. We present performance results on the moving MNIST and KITTI datasets with and without these modifications. We also study the effects of using the input and output gates coupled to the forget gate in contrast to leaving out the input and output gates entirely. We find that our architecture gives better next-frame prediction accuracy on these data sets while using a smaller parameter budget in comparison to the original Shi et al. (2015) convLSTM which uses elementwise multiply for peephole connections and vanilla convLSTM which lacks peephole connections.

2 Related Work

Recurrent neural networks (RNNs) process sequential data. Spatiotemporal datasets such as video are sequential datasets where both temporal and spatial information is related. Spatiotemporal prediction is a challenge that has received intense interest in deep learning over the last few years. Spatiotemporal prediction typically uses unsupervised learning. There are several mechanisms to predict future video frames using unsupervised learning such as (Srivastava et al., 2015; Shi et al., 2015; Lotter et al., 2017; Finn et al., 2016). However, the models use complex architectures and a large parameter budget.

The long short-term memory (LSTM) network, introduced by Hochreiter & Schmidhuber (1997), was the first gated RNN approach to mitigate the vanishing and/or exploding gradient problem that prevented RNNs from learning long-term dependencies. Hochreiter & Schmidhuber (1997) introduced a recurrent block that contained a memory cell which was updated by a constant-error-carousel (CEC) mechanism. The CEC was the key modification to mitigate the vanishing/exploding gradient problem. The main drawback of this first-generation LSTM was that if a continuous input stream was presented to the model long enough it could cause saturation of the output activation function due to the unlimited growth of the memory cell state values (Gers et al., 2000).

To address this problem, Gers et al. (2000) added a forget gate to the LSTM block. This design (standard LSTM (Gers et al., 2000)),which used a CEC, input, output, and forget gates, has since become known as the vanilla LSTM (Greff et al., 2017). The forget gate allowed the LSTM to update and/or reset its memory cell. This improved performance for learning long-term dependencies. However, this model also had weaknesses. First, the model had no direct connection from the memory state to the three gates. Thus, there was no control from the memory to the gates to assist in preventing the gradient from vanishing or exploding. Second, the CEC had no influence over the forget and input gates when the output gate was closed. This could harm the model due to the lack of primary information flow within the model (Gers et al., 2002). These problems were handled by adding peephole connections from the memory cell to each of the LSTM gates (Gers et al., 2002). Although the peephole generalization became a powerful model, there was a significant increase in the number of trainable parameters, the training time, and memory requirements.

There were attempts to design gated models to reduce the gate count while preserving learning power. Cho et al. (2014) proposed a gated recurrent unit (GRU) model. Instead of three gates, it used two: an update and a reset gate. The update gate combined the input gate and forget gate (Greff et al., 2017). The reset gate functioned as the output gate of the LSTM block (Greff et al., 2017). This GRU model eliminated the output activation function, memory unit, and CEC (Greff et al., 2017). The GRU yielded a reduction in trainable parameters compared with the vanilla LSTM. Zhou et al. (2016) used a single gate recurrent model called a minimal gated unit (MGU). Both models reduced the number of trainable parameters and gave results comparable to the LSTM (Chung et al., 2014). However, neither model preserved the CEC. This may lead to exploding and/or vanishing gradients. There was a study based on the LSTM model in (Gers et al., 2000) to examine the role and significance of each gate in the LSTM (Jozefowicz et al., 2015). This study showed that the most significant gate in the LSTM was the forget gate. However, it also showed that the forget gate needs other support to maintain its performance.

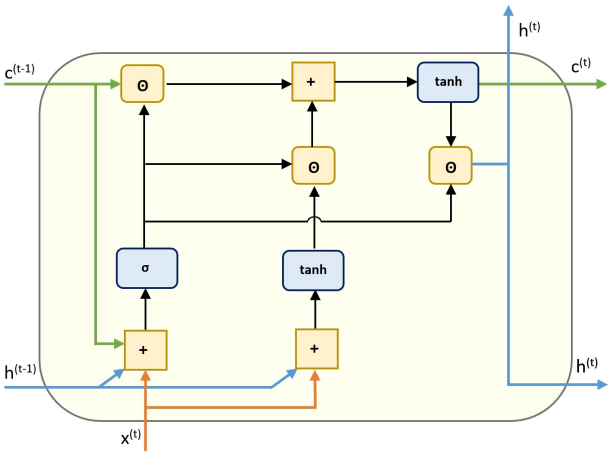

Figure 1: An unrolled block of the rgcLSTM. The single network gate (output indicated by σ) sends information to three locations which correspond to the outputs of the forget, input, and output gates of the vanilla LSTM.

While the above gated RNNs are designed to process sequential data, they must be augmented somehow to process spatial temporal data which contains images. The convLSTM operated on sequences of images. Shi et al. (2015) proposed a convolutional LSTM to enhance performance for spatiotemporal prediction. This model replaced the matrix multiplications (affine transformations) by convolutional operations for the input and recurrent input of each gate. The model achieved higher accuracy in comparison to the classical LSTM model. However, the number of parameters remained high because the peephole connections still used elementwise multiplication. In recent research, Lotter et al. (2017) showed how to build a predictive coding model using a convolutional LSTM architecture. The LSTM had three gates but no peephole connections. Greff et al. (2017) has termed this design the vanilla LSTM. The model achieved significant improvement in a predictive model as it did not need an encoder and decoder. However, the number of parameters was large and grew linearly with the number of layers in the model.

There were other attempts to design smaller recurrent gated models based on the standard LSTM. They were based on either removing one of the gates or the activation functions from the standard LSTM unit. Nonetheless, empirical analysis (Greff et al., 2017) compared these models and the standard LSTM. The conclusion found in Greff et al. (2017) was that these models had no significant improvement in either performance or training time. Moreover, these models had no significant reduction in trainable parameters. This empirical analysis (Greff et al., 2017) also stated that the critical components of the LSTM model were the forget gate and the output activation function.

Our new model is named the *reduced-gate, convolutional LSTM* (rgcLSTM). Based on the empirical results of Greff et al. (2017), our model preserves the critical components of the LSTM while removing parameter redundancy within the LSTM block. We use our model within a predictive coding framework introduced by Lotter et al. (2017) as a state-of-the-art approach for spatiotemporal prediction. The Lotter et al. (2017) model uses vanilla convLSTM modules (no peephole connections) within their predictive coding architecture. We replace those modules with rgcLSTM. Our model showed an efficiency comparable to Lotter et al. (2017) design. However, our rgcLSTM reduces the number of trainable parameters and memory requirements by about 40% and the training time by 50% which makes it more desirable to be implemented on hardware and trainable on low power devices.

3 Reduced-gate ConvLSTM Architecture

The proposed rgcLSTM model block appears in Figure 1. Since this is a convolutional LSTM, the information on each wire is a multi-channel image. The architecture has one trainable gated unit which we will call the forget gate or network gate. Because of its inputs, it is a forget gate, but since it is the only gate in the network it also makes sense to call it the network gate. The model also

preserves the cell memory state and keeps the CEC to avoid vanishing and/or exploding gradients. There is a peephole connection from the cell state to the network gate but we have converted its operator from elementwise multiplication to convolution. This reduces the learning capacity of the rgcLSTM in comparison to a full LSTM with elementwise peephole connections to all three gates. However, it preserves information needed to allow the memory state to exert control over the network gate. Also, our model retains the output activation function. Thus, our model preserves the critical components of the LSTM as stated by Greff et al. (2017) while removing much of the parameter redundancy in the LSTM unit. This gives in a significant reduction in the number of required trainable parameters, training time, memory, and hardware requirements compared to the vanilla convolutional LSTM and recurrent gated architectures. Furthermore, our rgcLSTM model preserves these model's prediction accuracy results.

During the forward pass within a block at time step t, the net input, $a^{(t)}$, to the single network gate $f^{(t)}$ is calculated by

$$a^{(t)} = [W_{fx}, U_{fh}, W_{fc}] * \left[x^{(t)}, h^{(t-1)}, c^{(t-1)}\right] + b_f. \tag{1}$$

The square brackets indicate that multi-channel images or filters with compatible dimensions are stacked on top of each other. $a^{(t)} \in \mathbb{R}^{\eta \times \upsilon \times n}$, and η, υ, and n are the width, the hight and the number of in-channels of the input $a^{(t)}$. $x^{(t)}$ is the input video frame at time t, $x^{(t)} \in \mathbb{R}^{\eta \times \upsilon \times \gamma}$ where η and υ are the width and height of the image, and γ is number of input image channels. $h^{(t-1)}$ is the output image stack of the block at time $t-1$, and $c^{(t-1)}$ is the image stack representing the block's internal state at time $t-1$. Both $h^{(t-1)}, c^{(t-1)} \in \mathbb{R}^{\eta \times \upsilon \times n}$ where η, υ, and n are the width, height, and number of core channels, respectively. W_{fx}, U_{fh}, and W_{fc} are convolution weight sets operating on their respective images or stacks. W_{fx} is $\in \mathbb{R}^{m \times m \times \gamma}$ and both U_{fh}, and W_{fc} are $\in \mathbb{R}^{m \times m \times \kappa}$. m is the convolution kernel width and height. For a given block, all kernels are the same size ($m \times m$). All three weight sets W_{fx}, U_{fh}, and W_{fc} and biases b_f are trainable. We will let $W_f = [W_{fx}, U_{fh}, W_{fc}] \in \mathbb{R}^{m \times m \times (\gamma + 2\kappa) \times n}$, where n is the number of output channels. Also, we let $I_f = \left[x^{(t)}, h^{(t-1)}, c^{(t-1)}\right] \in \mathbb{R}^{\eta \times \upsilon \times (\gamma + 2\kappa)}$. The convolution $W_f * I_f \in \mathbb{R}^{\eta \times \upsilon \times n}$. Finally, $b_f \in \mathbb{R}^{n \times 1}$ but is added to each channel in $W_f * I_f$ by broadcasting to the appropriate image channel. Note that the convolution operation between W_{fc} and $c^{(t-1)}$ represents a departure from Shi et al. (2015) where an elementwise multiply was used (in contrast to convolution).

The total number of trainable parameters for the network gate is

$$f^{\#}_{\text{gate}} = \left(m^2(\gamma + 2\kappa) + 1\right) \cdot n. \tag{2}$$

The network gate value, $f^{(t)}_{gate} \in \mathbb{R}^{\eta \times \upsilon \times n}$, is obtained by applying a pixelwise activation function G to the net input image using

$$f^{(t)}_{\text{gate}} = G(a^{(t)}). \tag{3}$$

Depending on the application, G can be either the logistic (σ) or hard sigmoid ($hardSig$) (Gulcehre et al., 2016). The pixel values of f^t will fall in the range $(0, 1)$ or $[0, 1]$, depending on which function is used. Using σ, the gate value f^t is calculated by

$$f^{(t)} = \sigma(W_f * I_f + b_f). \tag{4}$$

The input update (memory activation) uses a similar equation as for the network gate.

$$g^{(t)} = \tanh\left(W_g * I_g + b_g\right) \tag{5}$$

In the above, $W_g = [W_{gx}, U_{gh}] \in \mathbb{R}^{m \times m \times (\gamma + \kappa) \times n}$, and $I_g = \left[x^{(t)}, h^{(t-1)}\right] \in \mathbb{R}^{\eta \times \upsilon \times (\gamma + \kappa)}$. The number of core-channels in W_g is matching the number of core-channels of the W_f which make them computationally compatible. This approach is taken so that the dimension of $g^{(t)}_{\text{gate}} \in \mathbb{R}^{\eta \times \upsilon \times n}$ matches the dimension of $f^{(t)}_{\text{gate}}$. Similarly, the dimension of $b_g \in \mathbb{R}^{n \times 1}$ matches that of $b_f \in \mathbb{R}^{n \times 1}$. Finally, the number of trainable parameters for the input update is

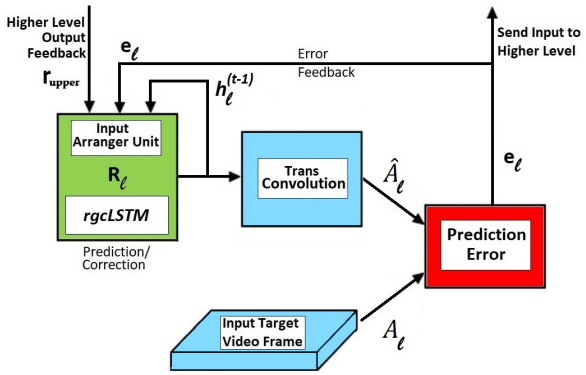

Figure 2: This is the bottom layer of the PredNet architecture with our rgcLSTM module substituted for the cLSTM in the original PredNet Lotter et al. (2017). The layer subscript l is used to signify that higher layers obey the same design, with the caveat that the input to higher layers is prediction error from the next lower layer.

$$g^{\#}_{\text{update}} = \left(m^2(\gamma + \kappa) + 1\right) \cdot n. \tag{6}$$

Eqns. 2 and 6 count the total number of trainable parameters for the rgcLSTM module/block, so the final count is given by

$$\text{rgcLSTM}^{\#}_{\text{block}} = f^{\#}_{\text{gate}} + g^{\#}_{\text{update}} \tag{7}$$

The final equations to complete the specification of the rgcLSTM are given below.

$$c^{(t)} = f^{(t)} \odot c^{(t-1)} + f^{(t)} \odot g^{(t)} \tag{8}$$

$$h^{(t)} = f^{(t)} \odot tanh(c^{(t)}) \tag{9}$$

The $\odot$ symbol denotes elementwise multiplication. κ is constrained to equal n so that the dimensions match for the elementwise multiplication operations.

4 Overall Architecture

We use our rgcLSTM module within the PredNet predictive coding framework of Lotter et al. (2017) and have named our variant Pred-rgcLSTM. The bottom layer of the architecture appears in Fig. 2. Our contribution lies in the design of the convLSTM block, where our rgcLSTM replaces the vanilla cLSTM. In this design, whether PreNet or Pred-rgcLSTM, learning only occurs within the gated recurrent blocks but is triggered by a prediction error signal. Source code for the model is available at `https://github.com/NellyElsayed/rgcLSTM`.

Although Fig. 2 shows the bottom layer (because the input is a video frame), we still include layer index subscripts, l, because the higher layers have similar structure (see Fig. 7). The prediction error module drives the gated-recurrent learning. It calculates the error between the input frame A_l and the predicted output frame $\hat{A}_l$. The $trans_l$ convolution module converts the output of the rgcLSTM, $h_l^{(t)}$, to $\hat{A}_l^{(t)}$ so the dimensions are compatible with A_l for pixelwise subtraction.

The prediction error module stacks the pixelwise difference between the predicted image frame $\hat{A}_l$ and the actual frame A_l by subtracting the two frames and applying the $ReLU$ as follows:

$$err1 = \text{ReLU}(\hat{A}_l - A_l) \tag{10}$$

$$err2 = \text{ReLU}(A_l - \hat{A}_l) \tag{11}$$

$$e_l = [err1, err2] \tag{12}$$

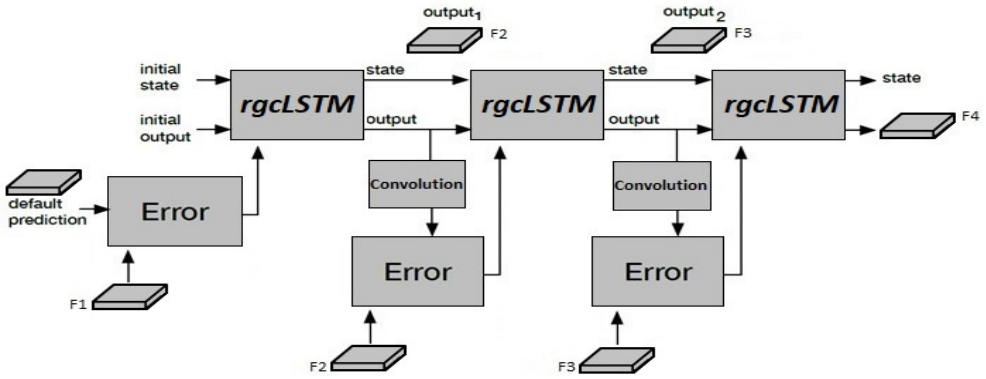

Figure 3: Three unrollment steps for the lowest layer of Pred-rgcLSTM using three input frames: F1, F2, and F3. F4 is the next-frame prediction.

e_l is a stack of the two error images from Eqns. 10 and 11. The E_{layer} feedback is sent to the rgcLSTM input arranger unit R_l and to the next higher layer R_{l+1} after applying convolutional downsampling for dimensions comparability with the next layer parameters dimensions. Inputs to the rgcLSTM module are its internal recurrent input and its error feedback. In the multi-layer case, the input contains an additional parameter which is the feedback from the directly higher rgcLSTM block, r_{upper}. The input arranger unit stacks these inputs to fit into the rgcLSTM block.

The rgcLSTM block learns the spatiotemporal changes using the training input data to predict future frames. The update process as a function of the sequential input within one layer of the proposed model as shown in Fig. 3. For illustration, we show three unrollment steps and start tracking the model from the leftmost (earliest) part of Fig. 3. In the beginning, the error module evaluates the prediction error by the difference between the assumed default prediction and the first input which is the first video frame. This prediction error then goes to the rgcLSTM module as an input. The rgcLSTM module processes this input, its initial state and output through its internal gates to produce the next predicted video frame. Next, the current output of the rgcLSTM module passes through the error module to evaluate the next prediction error which is forwarded to the next unfolded rgcLSTM module. At this time the prediction error acts as a correction factor for the rgcLSTM module which guides the rgcLSTM module to adjust its weights and state according to the current frame and the previous prediction of this frame. The process repeats until the final predicted frame.

Since our model targets spatiotemporal prediction, we used $hardSig$ as the recurrent activation function and $tanh$ as the output activation function. *hardSig* is calculated as

$$\text{hardSig}(x) = \max(\min(0.25\,x + 0.5, 1), 0). \tag{13}$$

We chose the $hardSig$ because it has shown better empirical results in our earlier experiments than the sigmoid in LSTM spatiotemporal prediction models Elsayed et al. (2018). The hard saturation may help it escape from local minima Gulcehre et al. (2016). The full multi-layer model architecture details are given in the appendix.

4.1 Keeping the Gate Output within a Functional Operating Range

From Fig. 1 we see that if the network gate value drops to zero, the rgcLSTM cell does an extreme reset: the memory state is forgotten, the current input is ignored, and the output is zero. In the predictive coding architecture, there are several factors that help keep the gate value within a functional range to avoid this. First, the input image pixel values are normalized to the range $[0, 1]$ and these become the values for A_0 in the lowest layer of the network (Eqns. 10 and 11). The values for $\hat{A}_l$ are also kept in the range $[0, 1]$. After going through the ReLU, the e_l values fall in the range $[0, 1]$. These form the inputs $x = e_l$ to the rgcLSTM. Within the rgcLSTM, the values of h and c (Fig. 1) are constrained to fall in the range $(-1, 1)$ because they are outputs of $\tanh$ or modulated $\tanh$. Finally, the network gate uses a hard sigmoid configured to be linear in the range $[-2, 2]$. The inputs to the hard sigmoid stay well within this range, thereby keeping the gate functional.

Table 1: Moving MNIST average elapsed training time and standard error (n=3).

Model	TrainTime	SE
PredNet	100.328	0.586532
Pred-rgcLSTM	**73.561**	**0.297411**

5 METHODS

Greff et al. (2017) showed that the vanilla LSTM (three gates, no peephole connections) exceeds the accuracy performance of other LSTM based approaches. Our experiments compare our rgcLSTM with the vanilla convLSTM. Moreover, we compare our model which uses the rgcLSTM with the most significant spatiotemporal prediction approaches that use recurrent gated units.

The model properties we consider are: existence of a memory cell, number of gates, and existence of peephole connections. The models we consider are our novel rgcLSTM, the vanilla LSTM (Gers et al., 2000), the peephole LSTM (PLSTM) (Gers et al., 2002), gated recurrent unit (GRU) (Cho et al., 2014), and the minimal gated unit (MGU) (Zhou et al., 2016). These models have been used on several approaches for video prediction. The next section compares our approach with the most significant of these previous approaches.

For training time comparisons, the models were trained on Intel(R) Core i7-7820HK CPU with 32GB memory and an NIVIDIA GeForce GXT 1080 graphics card for the first experiment. For the second experiment, both the models were trained on an Intel(R) Core i7-6700@4.00GHx8 processor with 64 GB memory and NIVIDIA GeForce GXT 980 Ti/PCIe/SSE2 graphics card.

6 RESULTS

We conducted two experiments on next-frame prediction of a spatiotemporal (video) sequence. The first experiment used gray-scale videos from the moving MNIST dataset which consists of MNIST digits moving in a 2D plane (Srivastava et al., 2015). The second experiment was applied to a RGB KITTI traffic video dataset. The moving MNIST dataset consists of $10,000$ different video sequences of two randomly selected moving digits from the MNIST data set. Each sequence is 20 frames long, with a frame size of 64x64 pixels. We divided the dataset into 6000 video sequences for training, 3000 for validation and 2000 for testing. The training process was completed in one epoch.

We trained our model using four rgcLSTM layers. The number of kernels in each layer were 1, 48, 96, and 192 respectively. We used the Adam optimizer (Kingma & Ba, 2014) with initial learning rate $\alpha = 0.001$, $\beta_1 = 0.9$ and $\beta_1 = 0.999$. We down-sampled the frame by factor of 2 moving upwards through the layers. We up-sampled the frame by factor of 2 moving down through the layers.

We compared the Pred-rgcLSTM (using rgcLSTM) and PredNet (using cLSTM). The training time (in minutes) comparison is shown in Table 1. Using the rgcLSTM block instead of the cLSTM reduced training time by approximately 45% for moving MNIST training using one epoch. Table 1 also shows the SE of model testing loss for both the cLSTM and our rgcLSTM models. The sample size of SE was $n = 3$ for each model. The SEs are very small compared to the mean differences.

Since we implemented our rgcLSTM model within the predictive-coding framework of Lotter et al. (2017), we named it Pred-rgcLSTM. We compared Pred-rgcLSTM with several other unsupervised recurrent gate-based prediction models (i.e. either LSTM or GRU based models) for the moving MNIST dataset. Performance measures included mean squared error (MSE) (Goodfellow et al., 2016), mean absolute error (MAE) (Goodfellow et al., 2016) and the structural similarity index (SSIM) (Wang et al., 2004) which is a measure for image quality similarity structure between predicted and target images. We also compared our model to PredNet which our model is based on but using the cLSTM. For the remaining models that we were unable to test due to hardware limits we obtained the results from their published work. All the models are convolutional variants.

Table 2: Moving MNIST performance comparison.

MODEL	MSE	MAE	SSIM
FCLSTM (Shi et al., 2015; Srivastava et al., 2015)	1.865	2.094	0.690
CDNA (Wang et al., 2018; Finn et al., 2016)	0.974	1.753	0.721
DFN (Wang et al., 2018; Jia et al., 2016)	0.890	1.728	0.726
VPN (Wang et al., 2018; Kalchbrenner et al., 2016)	0.641	1.310	0.870
ConvLSTM (Wang et al., 2018; Shi et al., 2015)	1.420	1.829	0.707
ConvGRU (Wang et al., 2018; Shi et al., 2017)	1.254	2.254	0.601
TrajGRU (Wang et al., 2018; Shi et al., 2017)	1.138	1.901	0.713
PredRNN++ (Wang et al., 2018)	0.465	1.068	0.898
PredRNN (Wang et al., 2017; 2018)	0.568	1.261	0.867
PredNet (Lotter et al., 2017)	0.011	0.049	0.915
Pred-rgcLSTM	**0.009**	**0.017**	**0.924**

Performance comparisons for moving MNIST are in Table 2. Our Pred-rgcLSTM shows a reduction of both the MSE and MAE compared to all of the other models. This includes FCLSTM (Srivastava et al., 2015), CDNA (Finn et al., 2016), DFN (Jia et al., 2016), VPN (Kalchbrenner et al., 2016), ConvLSTM (Shi et al., 2015), ConvGRU, TrajGRU, (Shi et al., 2017), PredRNN (Wang et al., 2017), PredRNN++ (Wang et al., 2018) and Prednet (Lotter et al., 2017). The main comparison is between our rgcLSTM block and the cLSTM block in the same model architecture. Our model also has the highest (best) structural similarity index measurement (SSIM) among the considered models.

Based on Eqn. 7, the number of trainable parameters for one rgcLSTM block is calculated as follows:

$$\begin{aligned}\text{rgcLSTM}^{\#}_{\text{block}} &= f^{\#}_{\text{gate}} + g^{\#}_{\text{update}} \\ &= \big(m^2(\gamma + 2\kappa) + 1\big) \cdot n + \big(m^2(\gamma + \kappa) + 1\big) \cdot n \\ &= 2 \times \left(m^2(\gamma + \frac{3}{2}\kappa) + 1\right) \cdot n\end{aligned} \tag{14}$$

For the cLSTM, the number of parameters is calculated by:

$$\begin{aligned}\text{cLSTM}^{\#}_{\text{block}} &= f^{\#}_{\text{gate}} + g^{\#}_{\text{update}} + i^{\#}_{\text{gate}} + o^{\#}_{\text{update}} \\ &= 4 \times \big(\big(m^2(\gamma + \kappa) + 1\big) \cdot n\big)\end{aligned} \tag{15}$$

Where $i^{\#}_{\text{gate}}$ and $o^{\#}_{\text{gate}}$ are the number of trainable parameters for the input and output gates of the cLSTM block. The multiplication by four is due to the input update activation, forget gate, input gate, and output gate that each have the same number of trainable parameters. Hence, the parameter number is reduced about by 40% in the rgcLSTM compared to the cLSTM. Additional parameters increasing the halving are due to the peephole connection which increases the size of the input to the rgcLSTM block.

Table 3: Moving MNIST number of trainable parameters in each model.

MODEL	TRAINING PARAMETERS
Model	**Training Parameters**
PredNet	6,909,834
ConvGRU	8,010,000
TrajGRU	4,770,000
ConvLSTM	7,585,296
FCLSTM	142,667,776
Pred-rgcLSTM	**4,316,235**

Using the Keras API, we counted the number of parameters used in each of the models in Table 3. We have also directly counted the parameters for PredNet and Pred-rgcLSTM (see appendix). Our model uses the fewest. Our model has fewer trainable parameters approximately by 40% than the

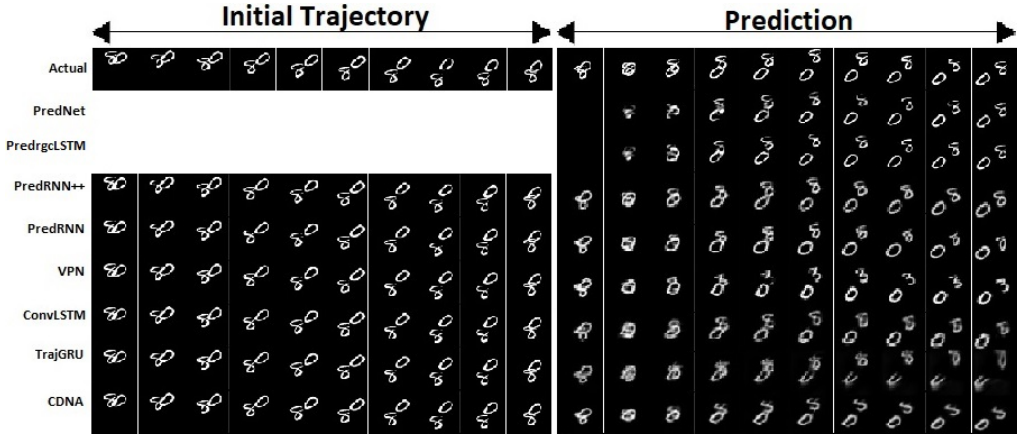

Figure 4: Visual results of Moving MNIST predictions after training based on our rgcLSTM and other models.

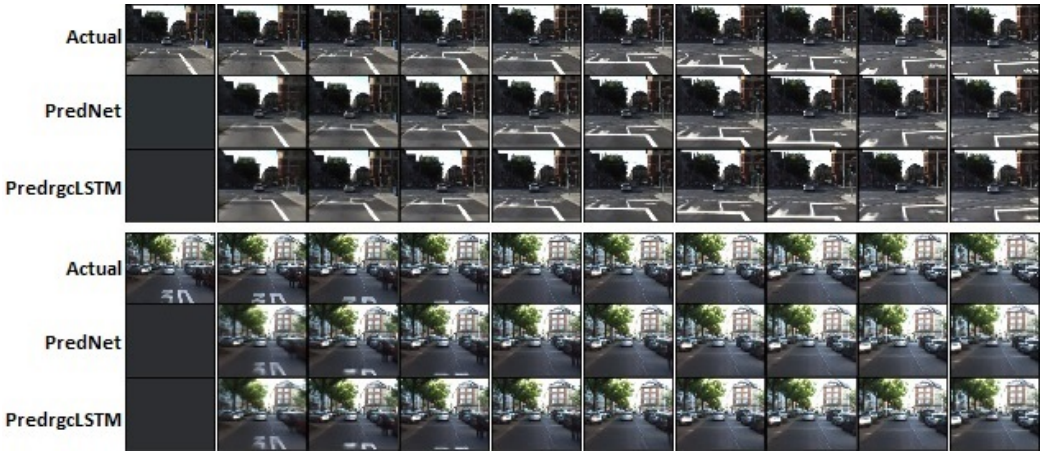

Figure 5: Next-frame prediction on the KITTI dataset. The predicted image is used as input for predicting the next frame.

ConvGRU and ConvLSTM respectively. Our Pred-rgcLSTM model that is based on rgcLSTM block has approximately 40% fewer trainable parameters that the original PredNet model that uses vanilla LSTM modules. The visual predictions for the moving MNIST data set appear in Fig. 4. In contrast to most of the models, both PredNet and our Pred-rgcLSTM do not require an initial input trajectory for predicting the next ten upcoming frames. It requires only the current frame and the current state of the trained LSTM to predict the next frame. Our rgcLSTM has comparable results with the other models and has better visual contours than the convolutional LSTM used in PredNet.

Our second experiment was applied to the KITTI dataset (Geiger et al., 2013). This dataset is a collection of real life traffic videos captured by a camera mounted on a car roof driving in an urban environment. The data is divided into three categories: city, residential, and road. Each category contained 57 videos. For training both the rgcLSTM and cLSTM we used the same initialization, training, validation and data down-sampling which was recommended by Lotter et al. (2017). After preprocessing, each frame size was 128x160 pixels. The length of the training dataset was approximately 41K frames. The visual testing results of our rgcLSTM and the cLSTM are shown in Fig. 5. This figure shows that the rgcLSTM has a visual improvement in precise prediction of object contours as compared with the the cLSTM with smaller training time and number of parameters.

The results of the KITTI dataset benchmark training time (in minutes), number of trainable parameters (# Params), mean squared error (MSE) (Goodfellow et al., 2016), mean absolute error (MAE) (Goodfellow et al., 2016), standard error (SE) (Goodfellow et al., 2016), and structural similarity

Table 4: Performance comparison on the KITTI traffic video dataset.

MODEL	TRAIN TIME	# Params	MSE	MAE	SE	SSIM	Memory
PredNet	230.982405	6,915,948	0.0035	0.030	0.000006	0.931	81.142 MB
Pred-rgcLSTM	**140.530364**	**4,320,273**	0.0035	0.030	**0.000005**	0.931	**50.697** MB

index measurement (SSIM) (Wang et al., 2004) of our rgcLSTM and cLSTM are shown in Table 4. The MSE, MAE, SE, and SSIM have a sample size of $n = 3$. In addition, Table 4 also shows that the memory required (MB) to save the trainable Pred-rgcLSTM model is smaller than the Prednet model. Both models have approximately the same MSE, MAE, and SSIM values. However, the rgcLSTM uses fewer trainable parameters and the training time is reduced by about $40\% - 50\%$ as in the moving MNIST (gray-scale) prediction task and a smaller standard error value (SE).

Fig. 6 shows the training and validation loss through 150 epochs of KITTI dataset training using both the proposed Pred-rgcLSTM and PredNet. The graphs show that the training and validation losses of both models is about the same. Therefore, the rgcLSTM is comparable with the cLSTM in regard to accuracy while using fewer trainable parameters, and less training time and memory.

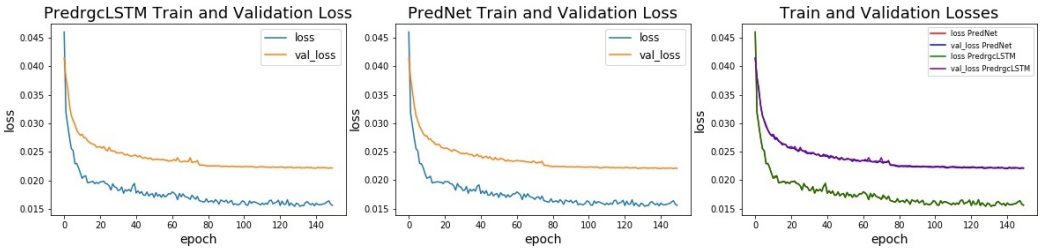

Figure 6: KITTI dataset training and validation losses for both Pred-rgcLSTM and PredNet. Left: Pred-rgcLSTM. Middle: Prednet. Right: Both graphs superimposed.

7 Conclusion

The novelty of our rgcLSTM architecture lies in the following features. First, there is one network gate that serves the function of the forget, input, and output gates. This reduces the gate parameter number to one third that of the vanilla LSTM. Second, in this reduced model there is still a peephole connection from the cell memory state to the network gate. Finally, the rgcLSTM uses a convolutional architecture and we have replaced the elementwise mulitiply operation originally used in Shi et al. (2015) with the convolution operation. This reduces the parameter number as well as being a more appropriate operator for spatio-temporal image processing.

The proposed rgcLSTM model outperforms the cLSTM and other gate-based recurrent architectures. Also, the model required fewer trainable parameters, memory, and training time. This was achieved by maintaining the critical components of the standard LSTM model, enhanced by the peephole connections and removing redundancy. These results make our rgcLSTM model attractive for future hardware implementation on small and mobile devices.

Remaining open questions for future work are: first, the model needs to be tested on more data sets; and, second, we need to rule out alternative explanations for the performance. For instance, the cLSTM may be suffering from a vanishing gradient issue that the rgcLSTM does not incur because it uses fewer parameters. These can be tested by introducing a ResNet architecture.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris

Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

Andre M Bastos, W Martin Usrey, Rick A Adams, George R Mangun, Pascal Fries, and Karl J Friston. Canonical microcircuits for predictive coding. *Neuron*, 76(4):695–711, 2012.

Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Nelly Elsayed, Anthony S. Maida, and Magdy Bayoumi. Empirical activation function effects on unsupervised convolutional LSTM learning. In *30th International Conference on Tools with Artificial Intelligence*. IEEE, 2018. in press.

Chelsea Finn, Ian Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction. In *Advances in neural information processing systems*, pp. 64–72, 2016.

Karl Friston. A theory of cortical responses. *Phil. Trans. R. Soc. B*, 360:815–836, 2005.

Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The KITTI dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.

Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with LSTM. *Neural Computation*, pp. 2451–2471, 2000.

Felix A Gers, Nicol N Schraudolph, and Jürgen Schmidhuber. Learning precise timing with LSTM recurrent networks. *Journal of machine learning research*, 3(Aug):115–143, 2002.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. LSTM: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232, 2017.

Caglar Gulcehre, Marcin Moczulski, Misha Denil, and Yoshua Bengio. Noisy activation functions. In *International Conference on Machine Learning*, pp. 3059–3068, 2016.

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

Roger A Horn. The hadamard product. In *Proc. Symp. Appl. Math*, volume 40, pp. 87–169, 1990.

Xu Jia, Bert De Brabandere, Tinne Tuytelaars, and Luc V Gool. Dynamic filter networks. In *Advances in Neural Information Processing Systems*, pp. 667–675, 2016.

Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In *International Conference on Machine Learning*, pp. 2342–2350, 2015.

Nal Kalchbrenner, Aaron van den Oord, Karen Simonyan, Ivo Danihelka, Oriol Vinyals, Alex Graves, and Koray Kavukcuoglu. Video pixel networks. *arXiv preprint arXiv:1610.00527*, 2016.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Keras: The Python Deep Learning library. Keras documentation, 2018. URL `https://keras.io/`.

William Lotter, Gabriel Kreiman, and David Cox. Deep predictive coding networks for video prediction and unsupervised learning. In *International Conference on Learning Representations*. Computational and Biological Learning Society, 2017.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS-W*, 2017.

Rajesh PN Rao and Dana H Ballard. Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. *Nature neuroscience*, 2(1):79, 1999.

X. Shi, Z. Chen, H. Wang, D. Yueng, W. Wong, and W. Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 28*, pp. 802–810. Curran Associates, Inc., 2015.

Xingjian Shi, Zhihan Gao, Leonard Lausen, Hao Wang, Dit-Yan Yeung, Wai-kin Wong, and Wang-chun Woo. Deep learning for precipitation nowcasting: A benchmark and a new model. In *Advances in Neural Information Processing Systems*, pp. 5617–5627, 2017.

Nitish Srivastava, Elman Mansimov, and Ruslan Salakhudinov. Unsupervised learning of video representations using LSTMs. In *International conference on machine learning*, pp. 843–852, 2015.

Yunbo Wang, Mingsheng Long, Jianmin Wang, Zhifeng Gao, and S Yu Philip. PredRNN: Recurrent neural networks for predictive learning using spatiotemporal LSTMs. In *Advances in Neural Information Processing Systems*, pp. 879–888, 2017.

Yunbo Wang, Zhifeng Gao, Mingsheng Long, Jianmin Wang, and Philip S Yu. PredRNN++: Towards a resolution of the deep-in-time dilemma in spatiotemporal predictive learning. *arXiv preprint arXiv:1804.06300*, 2018.

Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

Guo-Bing Zhou, Jianxin Wu, Chen-Lin Zhang, and Zhi-Hua Zhou. Minimal gated unit for recurrent neural networks. *International Journal of Automation and Computing*, 13(3):226–234, 2016.

Appendices

Table 5: Definition of terms

Symbol	Meaning
h_l	Eqn. 9.
e_l	Eqn. 12. $e_0 = x$.
c_l	Eqn. 8.
$r_{\text{upper}(l)}$	feedback from layer $l+1$.
R_l	dimension of gate input stack
Ra_l	dimension activation input stack
f_l	Eqn. 4.
g_l	Eqn. 5.
i_l	PredNet input gate for layer l.
o_l	PredNet output gate for layer l.
A_l	next frame input for layer l.
$\hat{A}_l$	predicted next frame input for layer l.
$downsample_l$	convolutional downsampling from layer l to $l+1$.
$biases_l$	total number of biases in layer l

We now explain the full multi-layer model shown in Fig. 7. Table 5 defines the terms used in the model. The input dimensions of each layer of the four-layered Pred-rgcLSTM and PredNet models are shown in Table 7. h_l is the recurrent output of the rgcLSTM or the cLSTM (which is equivalent to $h_{(t-1)}$), e_l is the error feedback (equivalent to $x_{(t)}$), c_l is the memory cell of the gated unit (equivalent to $c_{(t-1)}$), and $r_{\text{upper(l)}}$ is the higher-level feedback from layer $l+1$. R_l is the total dimension of each the forget gate $f^{(t)}$input stack and Ra_0 is the total dimension of the input stack to the activation unit $g^{(t)}$. Table 8 and Fig. 7 show the dimensions of the weights and biases in each layer of the the Pred-rgcLSTM and PredNet were for each layer index $l \in \{0, 1, 2, 3\}$. f_l, i_l, o_l are the forget, input and output gates, respectively. g_l is the activation unit. $downsample_l$ is the transition kernel from the lower layer to the higher layer. $\hat{A}_l$ is the convolution kernel applied to the rgcLSTM output or the cLSTM output. $Biases$ are the total number of biases in each layer (i.e. including the gate, activation, downsampling and the transition between the output of the rgcLSTM and the error module).

In Table 7, Table 8, and Fig. 7 we assume that the model input is a gray-scale image of size 64×64 pixels and has a one input channel depth.

For the purpose of counting trainable parameters, the values of m, γ, κ, and n from Section 3 are given in Table 6 for all four layers in the model.

Table 6: Dimension parameters for the gated module (either rgcLSTM or cLSTM) for each of the four layers for Moving MNIST.

Parameter	Layer$_0$	Layer$_1$	Layer$_2$	Layer$_3$
m	3	3	3	3
γ	50	192	384	384
κ	1	96	192	384
n	1	96	192	384

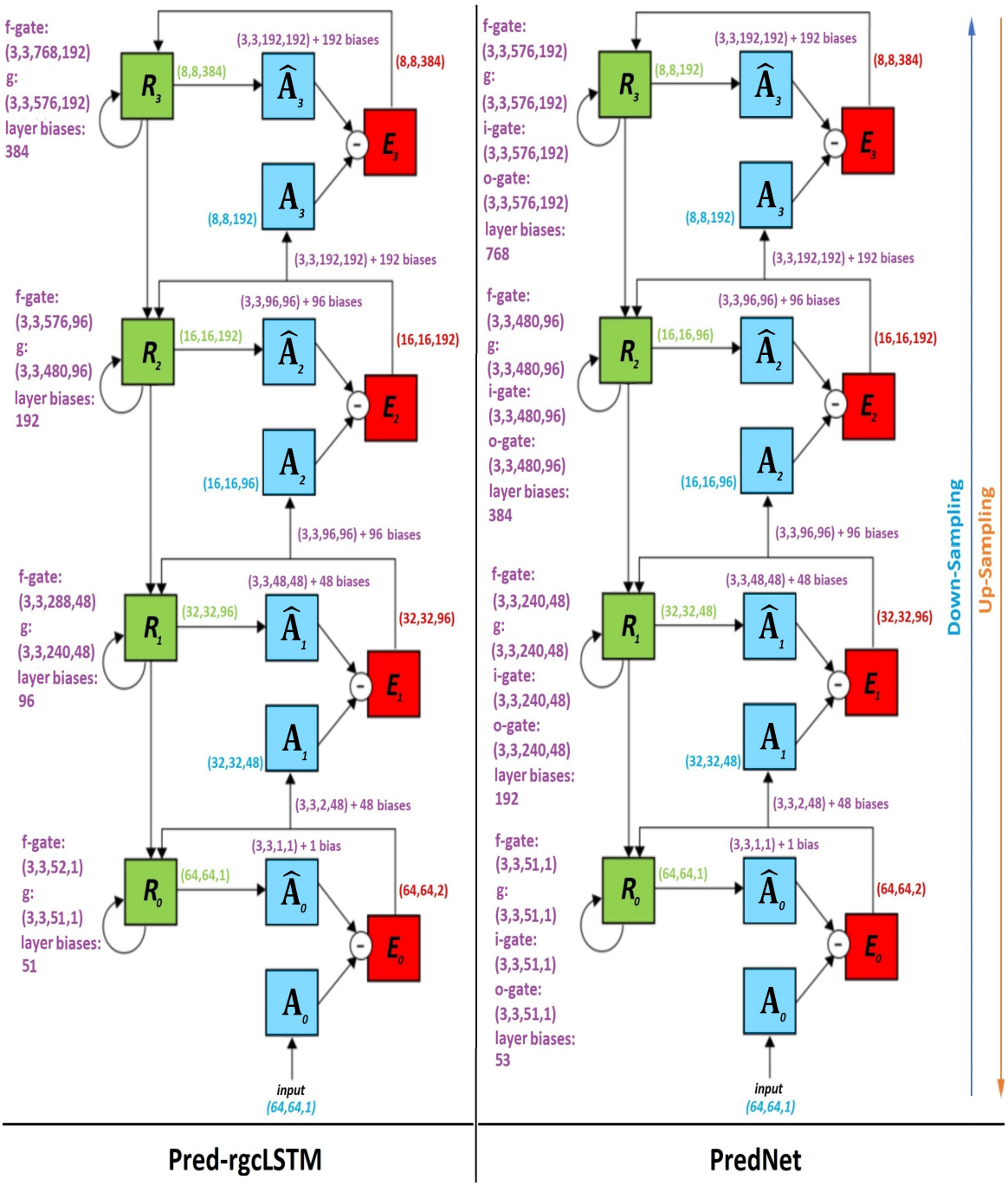

Figure 7: The weights of each component of both Pred-rgcLSTM and Prednet based on Lotter et al. (2017).

Table 7: Dimensions of input components to Pred-rgcLSTM and PredNet for Moving MNIST

	Dimensions	
Parameter	**Pred-rgcLSTM**	**PredNet**
h_0	(64, 64, 1)	(64, 64, 1)
e_0	(64, 64, 2)	(64, 64, 2)
c_0	(64, 64, 1)	N/A
$r_{\mathrm{upper}(0)}$	(64, 64, 48)	(64, 64, 48)
R_0 gate input stack	(64, 64, 51)	(64, 64, 51)
Ra_0 activation input stack	(64, 64, 52)	(64, 64, 51)
h_1	(32, 32, 48)	(32, 32, 48)
e_1	(32, 32, 96)	(32, 32, 96)
c_1	(32, 32, 48)	N/A
$r_{\mathrm{upper}(1)}$	(32, 32, 96)	(64, 64, 96)
R_1 input stack	(32, 32, 288)	(32, 32, 240)
Ra_1 activation input stack	(32, 32, 240)	(32, 32, 240)
h_2	(16, 16, 96)	(16, 16, 96)
e_2	(16, 16, 192)	(16, 16, 192)
c_2	(16, 16, 96)	N/A
$r_{\mathrm{upper}(2)}$	(16, 16, 192)	(16, 16, 192)
R_2 input stack	(16, 16, 576)	(16, 16, 480)
Ra_2 activation input stack	(16, 16, 480)	(16, 16, 480)
h_3	(8, 8, 192)	(8, 8, 192)
e_3	(8, 8, 384)	(8, 8, 384)
c_3	(8, 8, 192)	N/A
R_3 input stack	(8, 8, 768)	(8, 8, 576)
Ra_3 activation input stack	(8, 8, 576)	(8, 8, 576)

Table 8: Dimensions of kernel (weight) components of Pred-rgcLSTM and PredNet for Moving MNIST

	Dimensions	
Kernel	**Pred-rgcLSTM**	**PredNet**
f_0	(3, 3, 52, 1)	(3, 3, 51, 1)
g_0	(3, 3, 51, 1)	(3, 3, 51, 1)
i_0	N/A	(3, 3, 51, 1)
o_0	N/A	(3, 3, 51, 1)
$\hat{A}_0$	(3, 3, 1, 1)	(3, 3, 1, 1)
$downsample_0$	(3, 3, 2, 48)	(3, 3, 2, 48)
$biases_0$	51	53
f_1	(3, 3, 288, 48)	(3, 3, 240, 48)
g_1	(3, 3, 240, 48)	(3, 3, 240, 48)
i_1	N/A	(3, 3, 240, 48)
o_1	N/A	(3, 3, 240, 48)
$\hat{A}_1$	(3, 3, 48, 48)	(3, 3, 48, 48)
$downsample_1$	(3, 3, 96, 96)	(3, 3, 96, 96)
$biases_1$	240	336
f_2	(3, 3, 576, 96)	(3, 3, 480, 96)
g_2	(3, 3, 480, 96)	(3, 3, 480, 96)
i_2	N/A	(3, 3, 480, 96)
o_2	N/A	(3, 3, 480, 96)
$\hat{A}_2$	(3, 3, 96, 96)	(3, 3, 96, 96)
$downsample_2$	(3, 3, 192, 192)	(3, 3, 192, 192)
$biases_2$	480	672
f_3	(3, 3, 768, 192)	(3, 3, 576, 192)
g_3	(3, 3, 576, 192)	(3, 3, 576, 192)
i_3	N/A	(3, 3, 576, 192)
o_3	N/A	(3, 3, 576, 192)
$\hat{A}_3$	(3, 3, 192, 192)	(3, 3, 192, 192)
$biases_3$	576	960
Total trainable parameters	4,316,235	6,909,834