UNIVERSITY OF CALIFORNIA, SAN DIEGO
DEPARTMENT OF COGNITIVE SCIENCE

BACHELOR'S THESIS

Mental Simulation with Self-Supervised Spatiotemporal Learning

Author:
Kevin Tan

Supervisors:
Dr. Zhuowen Tu,
Dr. Virginia de Sa,
Kwonjoon Lee

Machine Learning, Perception, and Cognition Lab (mlPC)

June 2019

Abstract

Mental simulation – the capacity to imagine objects and scenes in order to make decisions, predictions, and inferences about the world – is a key feature of human cognition. Evidence from behavioral studies suggest that representations of visual imagery are spatial and sensitive to the causal structure of the world. Inspired by how humans anticipate future scenes, we aim to leverage state-of-the-art techniques in deep learning and computer vision to tackle the problem of spatiotemporal predictive learning in a self-supervised manner. We perform explorations across three architectural design choices: (i) the importance of 2D-convolution vs. 3D-convolution inside the cell of recurrent neural networks, (ii) the effectiveness of residual connections in stacked long short-term memory models for remembering spatial information over long time horizons, and (iii) the balance between l_1 norm and l_2 norm components in the objective function. Our extensive evaluations demonstrate that finetuning with residual connections achieves state-of-the-art performance on the Moving MNIST and KTH Action benchmark datasets. Potential application areas include weather forecasting, traffic flow prediction, and physical interaction simulation. Our source code is made available online [1].

[1] https://github.com/kevinstan/video_prediction

Acknowledgements

I would like to thank my research advisor, Professor Zhuowen Tu, for his guidance, support, and giving me a chance in the first place. I also thank Kwonjoon Lee, Yifan Xu, Wenlong Zhao, Haoyu Dong, and the rest of the Machine Learning, Perception, and Cognition Lab for valuable insights and discussions. I am indebted to Professor Virginia de Sa for accepting me as an undergraduate teaching assistant, for connecting me with IBM Research as a summer intern, and for her advice on applying to graduate school. I also thank Shuai Tang, who gave a great deal of assistance in running final experiments. I am grateful for the support from Shawn Hsu and Luca Pion-Tonachini during my time at SCCN. I appreciate the partial financial support from the Halıcıoğlu Data Science Institute Undergraduate Scholarship. Last, but far from least, I thank my girlfriend Emma Huang and my parents Hong and Jamie Tan for their love and support.

Contents

Introduction

Mental simulation is a phenomenon in which people can manipulate objects and scenes in their imagination in order to make predictions and inferences about the world. How do humans have the intuitive ability to perform mental simulation, and how can we develop machines capable of doing the same?

In recent years, artificial intelligence has seen tremendous progress in application areas such as image processing and natural language understanding due to exciting breakthroughs in deep learning. We've seen the emergence of computer programs capable of beating world-class chess players, driving autonomous vehicles, and generating their own creative writing. To say progress has been remarkable would be an understatement; yet, it still stands that the best existence proof of intelligence is in fact human intelligence. Today's machines, and in particular those that attempt to predict the visual future, still struggle to replicate the complexity and capabilities of visual mental imagery in humans.

Recent state-of-the-art models in spatiotemporal video prediction have been inspired by theories of mental representation. But the heated debate on representations of visual imagery (Kosslyn, 1994) has not yet reached a resolution since its inception in cognitive psychology. On the one hand, we have *analog*, or *depictive*, representations that make explicit and accessible the shape and the relations between shape and other percetual qualities (e.g. color and texture), as well as spatial relations among each point. On the other hand, we have *propositional* representations which posit that the format of mental images are symbolic similar to natural language descriptions or executable programs. Recent advancements such as ConvLSTM (Shi et al., 2015), PredRNN (Wang et al., 2017), and Eidetic 3D LSTM (Wang et al., 2019) are all inspired by representations of the first kind of representations: *analog*.

To tackle the problem of instilling machines with the capacity for visual imagination demands a crossdisciplinary approach from a variety of fields including computer science, machine learning, cognitive psychology, neuroscience, and philosophy. Doing so successfully will prove fruitful in a twofold manner: (1) by engineering more intelligent systems with visual understanding inspired by cognitive science, and (2) by shedding light on the computational algorithms of mental simulation and the debate on the nature of visual representations. The aim of this thesis is to study (1) as a means of achieving (2). We implement our approach of the Eidetic 3D LSTM with deep residual connections to study the problem of spatiotemporal predictive learning as a computational analog to mental simulation. We then explore a number of architectural design choices and evaluate our approach on two benchmark datasets.

The rest of this thesis is structured as follows. Chapter 2 begins with a review of modern deep neural networks and training techniques, followed by an overview of recent spatiotemporal predictive learning models, concluding with short digressions on deep residual learning and theories of mental imagery. Chapter 3 explicates the implementation details of our approach. Chapter 4 reports the experimental results on two benchmark datasets and gives a brief discussion and analysis. Finally, the conclusion summarizes the contributions and gives a brief overview of future directions.

Background and Related Work

2.1 Artificial Neural Networks

Artificial Neural Networks (ANNs), also known as *connectionist* systems, are computing architectures that inspired by, but not identical to, the biological neural networks in human and animal brains (McCulloch and Pitts, 1943). The connections between neurons are called *edges*, and the signal of the connection is represented by a signal of a real number in most implementations. The output of each artificial neuron is computed by a non-linear function of the sum of its inputs. Typically, edges are associated with a weight that adjusts as the learning procedure occurs, in which weights are updated to reflect the strength of the signal at a connection.

2.1.1 Architecture

Neural network models consist of two components: (i) the network architecture which defines how many neurons, how many layers, and how the neurons are connected, and (ii) the parameters, or weights, of the connections. *Deep* architectures refer to models constructed by a series of many layers in which each layer learns to transform its input data into a slightly more abstract and higher-level representation. Units are organized into L layers, where the first layer that gets fed the input is called the *input layer*, and the last layer yielding the predicted values is called the *output layer*. The intermediate layers are called *hidden layers*.

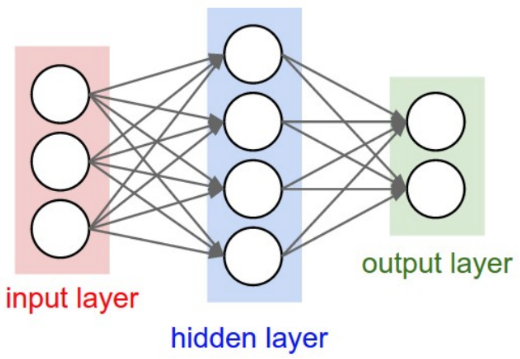


FIGURE 2.1: An ANN is an interconnected group of nodes, inspired by a simplification of neurons in the brain. Arrows represent the information flow in the feedforward direction from the input layer to output layer.

The activation of the layer $\mathbf{a}^{(l)}$ is expressed as:

$$\mathbf{z}^{(l)} = \mathbf{W}^{(l-1)}\mathbf{a}^{(l-1)} \quad \mathbf{a}^{(l)} = g\left(\mathbf{z}^{(l)}\right) \tag{2.1}$$

where $\mathbf{z}^{(l)}$ is the weighted sum of the inputs of each unit in the layer, upon which the non-linearity *activation function* g is applied, yielding $\mathbf{a}^{(l)}$ for layer l. Common non-linearities include sigmoid and hyperbolic tangent. It has been shown that with the correct choice of activation function, ANNs are able to approximate any continuous function in a compact subset of $\mathbb{R}^n$, making them universal approximators (Hornik, 1991).

2.1.2 Learning Procedure

Training examples $\mathbf{x}^{(\mathbf{i})}$ are fed forward layer by layer starting with the input layer. The *actual output* is compared with the *target output* via an *objective function* which measures the difference between them. The weights are learned through the back-propagation algorithm, which is an efficient way of computing gradients using the chain rule from calculus and dynamic programming techniques (Rumelhart, Hinton, and Williams, 1986). Training stops when the magnitude of error change reaches under a certain threshold known as the *stopping condition*.

Gradient Descent

To perform the backward error propogation in order to update the weights, the *gradient descent* algorithm or variants of it are commonly used. Gradient descent minimizes an objective function $J\left(\boldsymbol{\theta}, \mathbf{x}^{(i)}, \mathbf{y}^{(i)}\right)$ parameterized by $\boldsymbol{\theta} \in \mathbb{R}^d$ which measures the distance between predicted and true targets. *Batch* gradient descent computes the gradient of the objective function with respect to the parameters of the entire training dataset. In contrast, *Stochastic* gradient descent (SGD) computes the gradient for only one example and performs a parameter update for each training example one at a time.

The update rule is given by:

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} J\left(\boldsymbol{\theta}, \mathbf{x}^{(i)}, \mathbf{y}^{(i)}\right) \tag{2.2}$$

where

$$\nabla_{\boldsymbol{\theta}} J\left(\boldsymbol{\theta}, \mathbf{x}^{(i)}, \mathbf{y}^{(i)}\right) = \begin{bmatrix} \frac{\partial J\left(\boldsymbol{\theta}, \mathbf{x}^{(i)}, \mathbf{y}^{(i)}\right)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J\left(\boldsymbol{\theta}, \mathbf{x}^{(i)}, \mathbf{y}^{(i)}\right)}{\partial \theta_d} \end{bmatrix} \tag{2.3}$$

and $\eta \in \mathbb{R}$ is known as the *learning rate*. It is common in practice to use *Mini-Batch* gradient descent, which takes a mini-batch of n examples and computes the gradient from those examples:

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} J\left(\boldsymbol{\theta}, \mathbf{x}^{(i:i+n)}, \mathbf{y}^{(i:i+n)}\right) \tag{2.4}$$

Adam Optimizer

Adaptive Moment Estimation (Adam) (Kingma and Ba, 2014) is an optimization algorithm (similar to RMSProp (Tieleman and Hinton, 2012) and Adadelta (Zeiler, 2012)) that computes adaptive learning rates for each parameter, and has been shown to work well in practice and empirically outperforms other adaptive learning algorithms.

Adam stores an exponentially decaying average of past squared gradients v_t but also an exponentially decaying average of past gradients m_t, as follows:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1)\, g_t \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2)\, g_t^2 \end{aligned} \tag{2.5}$$

where β_1 is the decaying coefficient for the running average of the gradient (typically 0.9) and β_2 is the decaying coefficient for the running average of the squared gradient (typically 0.999).

Then the bias-corrected estimates are computed as:

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1-\beta_2^t} \end{aligned} \tag{2.6}$$

to yield the Adam update rule:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t \tag{2.7}$$

Batch Normalization

Batch normalization is a technique for accelerating training and improving the speed, performance, and stability of ANNs. It was initially proposed to solve internal covariate shift (Ioffe and Szegedy, 2015). The central idea is normalize the input to a layer by adjusting and scaling the activations. Using a mini-batch of examples, a neuron x in the pre-activation output of a layer is normalized by:

$$\hat{x} = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x]}} \tag{2.8}$$

where the resultant mean and variance are zero and one, respectively.

In order not to lose the representational power of the network due to normalization, two more parameters are introduced (for each neuron in the network): γ, β that can "undo" the normalization:

$$y = \gamma\hat{x} + \beta \tag{2.9}$$

As a result, batch normalization introduces many benefits: allowing a substantially larger learning rate for training without exploding gradients, reducing the need for dropout layers by helping regularize the network, and being more robust to different initialization schemes.

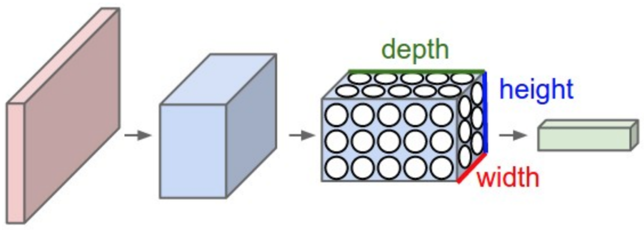


FIGURE 2.2: A ConvNet arranges its neurons in three dimensions (width, height, depth) as visualized in one of the layers. Every layer of a ConvNet transforms its 3D input volume to a 3D output volume of neuron activations. The red layer denotes the input image, with width and height as its dimensions, and depth of 3 corresponding to red, green, blue (RGB) channels.

2.2 Convolutional Neural Networks

Convolutional Neural Networks (CNN) (Lecun et al., 1998) are a special class of deep ANNs that contain one or more convolutional layers. They are designed to work well for image processing based on their shared-weights architecture and translational invariance characteristics.

Contrary to traditional multilayer perceptrons (MLP), CNNs have the following distinguishing features[1]:

1. **Local connectivity**: Since it is impractical to connect neurons in each layer to all the neurons in the previous layer's output when working with high-dimensional inputs such as images, neurons are instead connected to only local regions. The local spatial size of the receptive field is a hyperparameter, and the depth is always equal to the depth of the input volume.

2. **Shared weights**: To control the number of learnable weights, CNNs make the assumption that if it is useful to compute features learned at one spatial location (x_1, y_1), then it is also useful to compute features learned at another spatial location (x_2, y_2).

There are three main types of layers used to build ConvNet architectures: Convolutional Layer, Pooling Layer, and Fully-Connected Layer. In this thesis, we focus on the Convolutional Layer, and – in particular – the differences between 2D and 3D convolutions.

2.2.1 Convolutional Layer

The convolutional layer is the core building block of a CNN that does most of the computation. The layer's parameters consist of a set of learnable kernels which typically have a small receptive field emulating the response of an individual neuron to visual stimuli. The kernels extend the full depth of the input volume. During the forward pass, each kernel is *convolved* across the input, computing the dot product and producing feature maps of that kernel. As a result, the kernel learns to activate when detecting a particular feature, invariant of the spatial location in the input.

[1]http://cs231n.github.io/convolutional-networks/

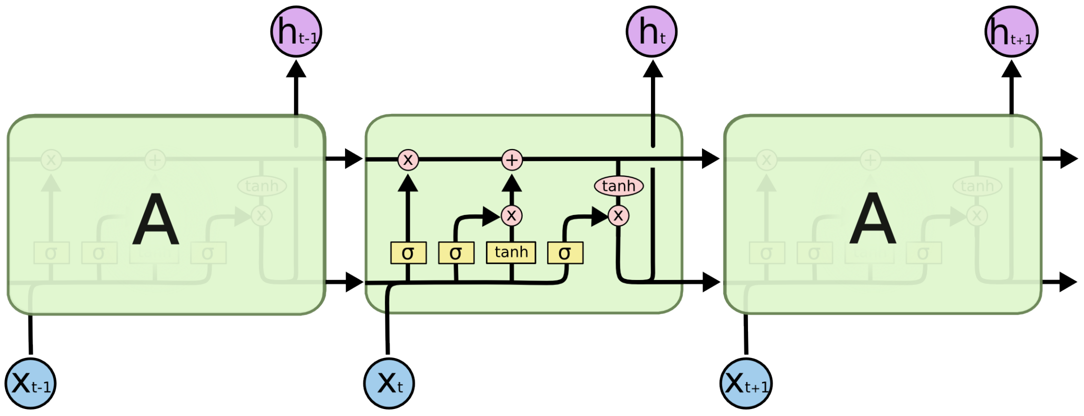


FIGURE 2.3: The repeating module in an LSTM contains four interacting components: cell, input, output, and forget. Yellow boxes denote neural network layers, pink circles denote pointwise operations, and black arrows denote the flow of vector information. Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

2.3 Long Short-Term Memory for Sequence Modeling

Long Short-Term Memory (LSTM) is a special case of a recurrent neural network (RNN) architecture that has proven to be more stable and powerful in modeling dependencies over long-range temporal horizons (Graves, 2013; Hochreiter and Schmidhuber, 1997). LSTM networks were developed to deal with the exploding and vanishing gradient problem that can commonly be encountered when training traditional RNNs; consequently, they are well-suited for sequence modeling tasks based on time series data. RNN architectures have the property of forming a chain of repeating modules of neural networks. In LSTM networks, the recurrent module is composed of a cell, an input gate, an output gate, and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the information flow as input to and output from the cell.

The major innovation of LSTM is the innovation of the memory cell c_t that accumulates state information over time. New inputs will be accumulated into the cell state if the input gate i_t is activated. The previous cell state c_{t-1} could be "forgotten" if the forget gate f_t is activated. The updated cell output is c_t is controlled by output gate o_t to determine whether it is propogated into the hidden state h_t. Multiple LSTMs can be stacked in the vertical direction and temporally concatenated to form more complex architectures.

The key equations are given by (2.10) where '$\circ$' denotes the Hadamard product:

$$\begin{aligned}
i_t &= \sigma\left(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci} \circ c_{t-1} + b_i\right) \\
f_t &= \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf} \circ c_{t-1} + b_f\right) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \tanh\left(W_{xc}x_t + W_{hc}h_{t-1} + b_c\right) \\
o_t &= \sigma\left(W_{xo}x_t + W_{ho}h_{t-1} + W_{co} \circ c_t + b_o\right) \\
h_t &= o_t \circ \tanh\left(c_t\right)
\end{aligned} \tag{2.10}$$

2.4 Spatiotemporal Predictive Learning

Shi et al., 2015 has proposed the following scenario for spatiotemporal predictinve learning. Suppose we are observing a video clip represented by P measurements over time, where each measurement (e.g. RGB channel) is recorded at all locations in a spatial region represented by an MxN grid representing the dimensions of a video frame. From the spatial view, the observation of these P measurements at any point in time can be represented by the tensor $\mathcal{X} \in \mathbb{R}^{PxMxN}$. From the temporal view, the observations over T steps form a sequence of tensors $\mathcal{X}_1, \mathcal{X}_2, ..., \mathcal{X}_T$. From a machine learning perspective, the problem of spatiotemporal prediction can be seen as a sequence forecasting problem. The goal is to predict the most probable length-K sequence in the future given the previous length-J sequence including the observation at the current time step:

$$\widehat{\mathcal{X}}_{t+1}, \ldots, \widehat{\mathcal{X}}_{t+K} = \underset{\mathcal{X}_{t+1},\ldots,\mathcal{X}_{t+K}}{\arg\max}\; p\left(\mathcal{X}_{t+1}, \ldots, \mathcal{X}_{t+K} | \mathcal{X}_{t-J+1}, \ldots, \mathcal{X}_t\right) \tag{2.11}$$

In the context of video generation, the measurements are three RGB channels, and the observation at each time step is a 3D tensor (H, W, C) where H and W are the height and width of the frame respectively, and C is the number of channels.

2.4.1 Convolutional LSTM

The Convolutional LSTM (ConvLSTM) (Shi et al., 2015) is able to model spatiotemporal sequences by simultaneously encoding the spatial information into tensors, overcoming the limitation of standard LSTM where spatial memory is lost. In the context of ConvLSTMs, all the inputs $\mathcal{X}_1, ..., \mathcal{X}_t$, cell outputs $\mathcal{C}_1, ..., \mathcal{C}_t$, hidden states $\mathcal{H}_1, ..., \mathcal{H}_t$, and gates i_t, f_t, g_t, and o_t are 3D tensors in $\mathbb{R}^{PxMxN}$, where the first dimension represents either the number of measurements (in the case of inputs) or the number of feature maps (for intermediate representations), and the last two dimensions represent the spatial dimensions of the frame (e.g. M rows and N columns). One may imagine inputs and states to be vectors standing on a spatial grid (2.4). The future cell state of a particular cell in the MxN grid is then determined by its inputs at the current time step and past states of local neighbors. Intuitively, a ConvLSTM with a larger transition kernel should be able to capture faster motions while one with a smaller kernel can capture slower motions (Shi et al., 2015).

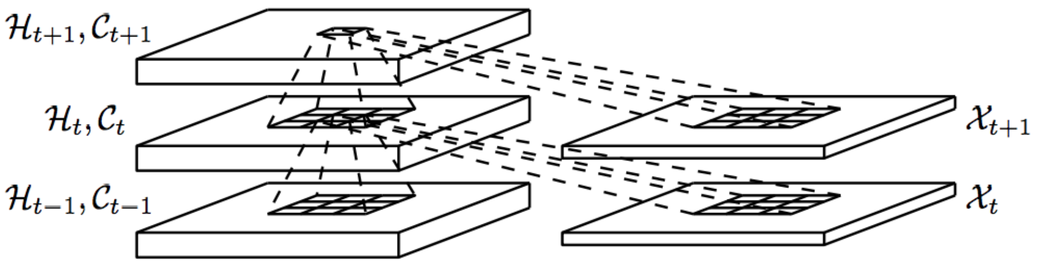

$\mathcal{H}_{t+1}, \mathcal{C}_{t+1}$
$\mathcal{H}_t, \mathcal{C}_t$
$\mathcal{H}_{t-1}, \mathcal{C}_{t-1}$
$\mathcal{X}_{t+1}$
$\mathcal{X}_t$

FIGURE 2.4: The internal structure of ConvLSTM.

The ConvLSTM network architecture adopts the encoder-decoder RNN scheme (Sutskever, Vinyals, and Le, 2014). For a ConvLSTM network with 4 layers, input frames are fed into the first layer and the future predicted frame is generated at the fourth layer. In this process, spatial representations are encoded layer-by-layer with hidden memory cells passed from the first layer to the top layer in the vertical direction. However, this layer-independent memory mechanism limits spatial memory to interact in the temporal direction.

2.4.2 PredRNN

PredRNN (Wang et al., 2017) is a slight modification to the ConvLSTM architecture that incorportes memory state flow in zigzag manner – from the final memory state of the previous time step as input into the first memory state of the current time step (2.5). Similar to (2.10), the key equations of PredRNN are given as follows:

$$\begin{aligned}
g_t &= \tanh\left(\mathcal{W}_{xg} * \mathcal{X}_t + \mathcal{W}_{hg} * \mathcal{H}_{t-1}^l + b_g\right)\\
i_t &= \sigma\left(\mathcal{W}_{xi} * \mathcal{X}_t + \mathcal{W}_{hi} * \mathcal{H}_{t-1}^l + b_i\right)\\
f_t &= \sigma\left(\mathcal{W}_{xf} * \mathcal{X}_t + \mathcal{W}_{hf} * \mathcal{H}_{t-1}^l + b_f\right)\\
\mathcal{C}_t^l &= f_t \odot \mathcal{C}_{t-1}^l + i_t \odot g_t\\
g_t' &= \tanh\left(\mathcal{W}_{xg}' * \mathcal{X}_t + \mathcal{W}_{mg} * \mathcal{M}_t^{l-1} + b_g'\right)\\
i_t' &= \sigma\left(\mathcal{W}_{xi}' * \mathcal{X}_t + \mathcal{W}_{mi} * \mathcal{M}_t^{l-1} + b_i'\right)\\
f_t' &= \sigma\left(\mathcal{W}_{xf}' * \mathcal{X}_t + \mathcal{W}_{mf} * \mathcal{M}_t^{l-1} + b_f'\right)\\
\mathcal{M}_t^l &= f_t' \odot \mathcal{M}_t^{l-1} + i_t' \odot g_t'\\
o_t &= \sigma\left(\mathcal{W}_{xo} * \mathcal{X}_t + \mathcal{W}_{ho} * \mathcal{H}_{t-1}^l + \mathcal{W}_{co} * \mathcal{C}_t^l + \mathcal{W}_{mo} * \mathcal{M}_t^l + b_o\right)\\
\mathcal{H}_t^l &= o_t \odot \tanh\left(\mathcal{W}_{1\times1} * \left[\mathcal{C}_t^l, \mathcal{M}_t^l\right]\right)
\end{aligned} \tag{2.12}$$

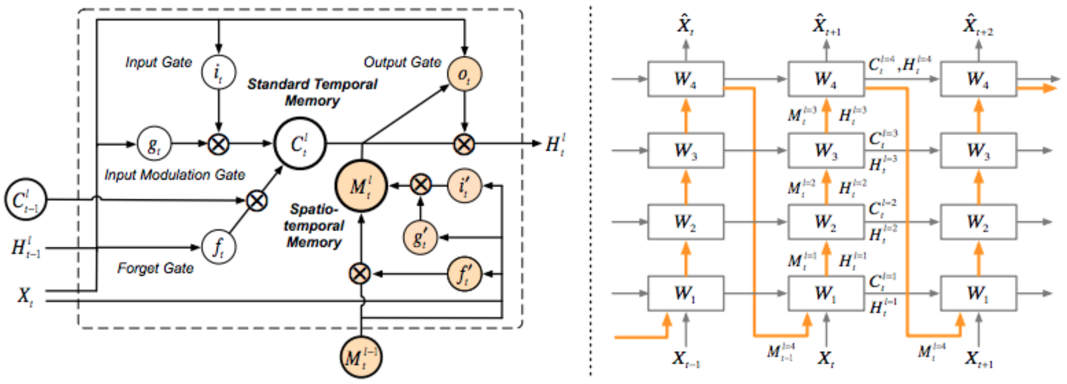


FIGURE 2.5: Spatiotemporal LSTM (ST-LSTM) (Left) and PredRNN (Right). The orange circles in ST-LSTM show the differences with ConvLSTM. The orange arrows in PredRNN show the transition path of spatiotemporal memory.

A novel spatiotemporal memory state $\mathcal{M}_t^l$ is introduced which is passed through subsequent stacked layers at the same time step from bottom to top. Importantly, the bottom layer at time t where $l = 1$ is described by $\mathcal{M}_t^{l=1} = \mathcal{M}_{t-1}^L$ where layers are indexed by $1, ..., L$. The hidden states apply a $1x1$ convolution layer for dimension reduction to make them the same size as the memory states. The resulting architecture effectively models shape deformations and motion trajectories in spatiotemporal sequences.

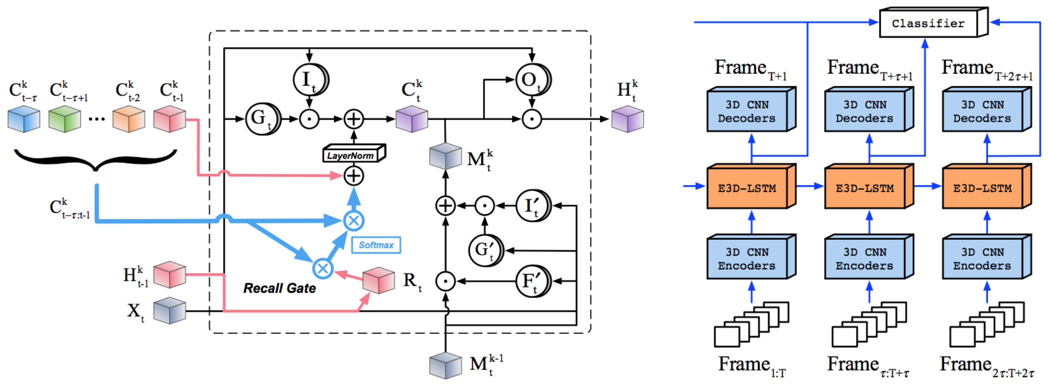
2.4.3 Eidetic 3D LSTM

The Eidetic 3D LSTM (E3D-LSTM) (Wang et al., 2019) proposes a deeper integration of 3D convolution inside the LSTM unit in order to incorporate convolutional features into the recurrent state transition over time. In this model, a consecutive of T input frames are first encoded by a few layers of 3D-Conv to obtain high-dimensional feature maps. The 3D feature maps are fed as input to the bottom layer of a 4-layer E3D-LSTM to model long-term spatiotemporal interaction. The final hidden states are decoded by a number of stacked 3D-Conv layers to get the predicted frames.

Crucially, E3D-LSTM introduces the "eidetic" attention mechanism denoted by the RECALL function:

$$\begin{aligned}
\mathcal{R}_t &= \sigma\left(W_{xr} * \mathcal{X}_t + W_{hr} * \mathcal{H}_{t-1}^k + b_r\right) \\
\mathcal{I}_t &= \sigma\left(W_{xi} * \mathcal{X}_t + W_{hi} * \mathcal{H}_{t-1}^k + b_i\right) \\
\mathcal{G}_t &= \tanh\left(W_{xg} * \mathcal{X}_t + W_{hg} * \mathcal{H}_{t-1}^k + b_g\right) \\
\text{RECALL}\left(\mathcal{R}_t, \mathcal{C}_{t-\tau:t-1}^k\right) &= \text{softmax}\left(\mathcal{R}_t \cdot \left(\mathcal{C}_{t-\tau:t-1}^k\right)^\top\right) \cdot \mathcal{C}_{t-\tau:t-1}^k \\
\mathcal{C}_t^k &= \mathcal{I}_t \odot \mathcal{G}_t + \text{ LayerNorm }\left(\mathcal{C}_{t-1}^k + \text{RECALL}\left(\mathcal{R}_t, \mathcal{C}_{t-\tau:t-1}^k\right)\right)
\end{aligned} \tag{2.13}$$

where σ is the sigmoid function, $*$ is the 3D-Conv operation, $\odot$ is the Hadamard product, $\cdot$ is the matrix product after reshaping the recall gate R_t and memory states $\mathcal{C}_{t-\tau:t-1}^k$ into $\mathbb{R}^{THW\times C}$ and $\mathbb{R}^{\tau THW\times C}$ respectively, and τ is the number of memory states that are concatenated along the temporal dimension.



(A) (B)

FIGURE 2.6: (A): E3D-LSTM Recurrent Unit. Cubes denote higher-dimensional hidden states and memory states. (B): E3D-LSTM Network with 3D-encoders and 3D-decoders.

2.5 Deep Residual Learning

A *residual neural network* (He et al., 2015) is a type of ANN inspired by constructs known as pyramidal cells in the cerebral cortex. Traditional ANNs contain layers that carry information in a sequential manner from layer to layer. In contrast, residual neural networks are built by using skip-connections to jump over layers. While any layer can be connected to another layer in a feedforward fashion in theory, it is often seen that double or triple skip connections that contain nonlinearities with batch normalization in between works best in practice. Residual connections have the option to be implemented with an additional weight matrix for learning the skip weights, or they can simply be tensor operations from layer to layer.

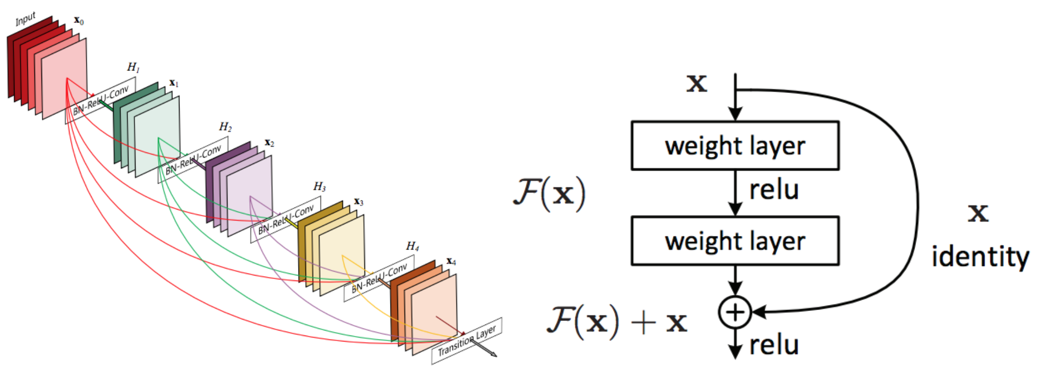


FIGURE 2.7: *Left:* Dense Convolutional Network (DenseNet) architecture, connecting each layer to every other layer in a feedforward fashion. Example of deep residual learning (Huang, Liu, and Weinberger, 2016). *Right:* The building block of residual learning (He et al., 2015).

2.6 Mental Representations of Visual Imagery

With the rise of behaviorism, the study of mental imagery declined through the middle part of the 20th century. However in 1971, Shepard and Metzler (S&M) famously introduced the concept of mental rotation into the cognitive science literature – in what has now become known as one of the most influential experiments of the field (Shepard and Metzler, 1971). The experiment was to tell as quickly as possible whether two cube-shaped figures were either identical or mirror images. S&M hypothesized that subjects form mental three-dimensional representations of one of the depicted objects and rotated the representation in imagination to match the other. Their experimental results directly supported this idea, by demonstrating that the time taken to respond to both objects was directly proportional to the increase in angular difference between them.

In support of the idea that *thinking is essentially computation*, these results suggested that subjects run algorithmic mental processes that require longer compute time for increasingly computationally hard problems. Crucially, Shepard interpreted the results to support the doctrine of analog and intrinsically spatial mental processes underlying thinking, advocating for what he called "second order isomorphism(s)" between objects and their mental images. Further research on mental imagery (Kosslyn, 1994) gave rise to one of the most famous debates in cognitive

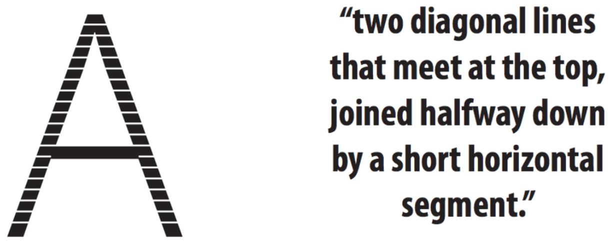

“two diagonal lines that meet at the top, joined halfway down by a short horizontal segment.”

FIGURE 2.8: *Left:* Analog representation of the letter "A". *Right:* Propositional representation of the letter "A".

psychology about whether mental imagery is either (a) analog/spatial/depictive (i.e. a literal drawing or functional image like an arrangement of pixels in computer memory), or (b) propositional (i.e. symbolic akin to natural or programming languages). Nevertheless, the imagery debate has not yet been official resolved, but it stands clear that mental representations are inherently spatial and subject to the causal structure of the world.

Implementation

This section presents the implementation details for our approach: Eidetic 3D LSTM with deep residual connections. We make explorations across three design choices:

1. 2D-Conv vs. 3D-Conv inside RNN cell
2. Residual connections architecture (addition vs. concatenation)
3. Balance between l_1 and l_2 norms in the loss function

Software and Hardware All experiments are conducted using TensorFlow (Abadi et al., 2016) v1.9.0 and trained with the ADAM optimizer. Experimental hardware consists of 4 NVIDIA TITAN X (Pascal) 12GB GPUs (with CUDA v9.0 and Driver 390.116) to perform parallel computation. To ensure fair comparisons, all models are trained with a comparable number of parameters and apply the same scheduled sampling strategy (Bengio et al., 2015) to improve the training efficiency of recurrent neural nets.

3.1 E3D-LSTM Hyperparameters

4 E3D-LSTMs are stacked in the vertical direction, leaving out 3D-CNN encoders for this task. Integrated 3D-Conv operators are composed of a $2 \times 5 \times 5$ (time $\times$ width $\times$ height) kernel to retain the shape of the hidden states across time. The number of hidden output channels is 64. The temporal window length is set to 1 so there is a single overlapping frame during convolution over consecutive time steps. A single 3D-Conv decoder is used to map hidden states to generated frames. Additional comparisons are made between the number of stacked LSTM layers (2 or 4) and between 2D-Conv and 3D-Conv operations inside the recurrent cell. For 2D-Conv experiments, the integrated 2D-conv operator is composed of a $1 \times 5 \times 5$ kernel such that there is no overlapping frame in the temporal direction over consecutive convolutions.

3.2 Residuals Setup

Residual connections are integrated across the 4 layers such that the output of the 1st layer plus the output of the 2nd layer is fed as input to the 3rd layer, and the output of the 3rd layer plus the output of the 4th layer is fed as the input to the 3D-Conv decoder. Importantly, residual connections do not introduce any additional learnable parameters. We find that the addition operation as opposed to concatenation shows better empirical performance. Note that addition can be seen as a special case of concatenation where the 3D-Conv decoder selects from each layer of the stacked LSTM with equal weight.

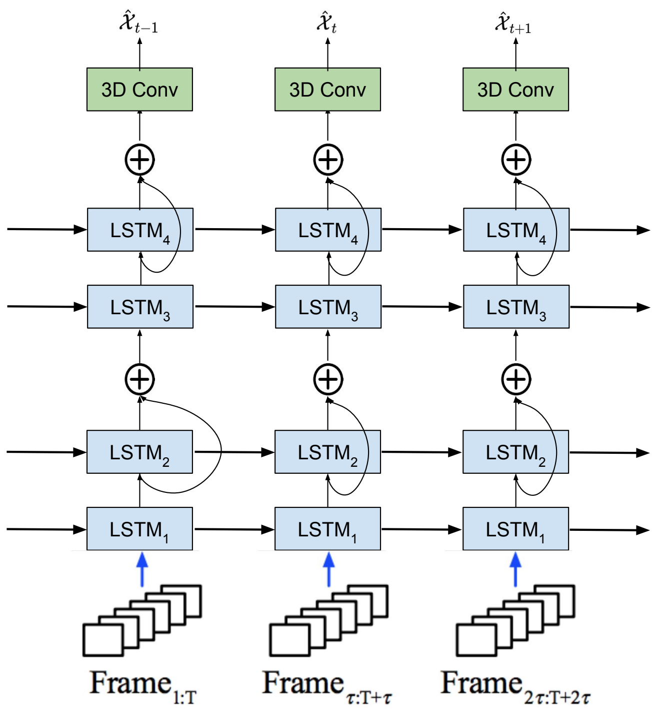


FIGURE 3.1: Architectural design of our approach: E3D-LSTM with deep residual connections. 4 E3D-LSTMs are stacked in the vertical direction with residual connections between before layer 3 and before the decoder. $\oplus$ denotes the addition operator.

3.3 Loss Function

The default objective function is an equally weighted sum of the l_1 and l_2 norms to minimize the loss over every pixel in the frame. We use the `tf.nn.l2_loss(...)` and `tf.reduce_sum(tf.abs(...))` as the l_2 and l_1 components, respectively. Additional comparisons are made between using only l_1 loss and only l_2 loss.

Experiments

4.1 Experiment 1: Moving MNIST Dataset

Dataset. The Moving MNIST dataset is constructed by randomly sampling two digits from the original MNIST dataset and making them float and bounce against the boundaries of the frame with a constant velocity and angle inside a black canvas of 64×64 pixels. The whole dataset consists of 10,000 sequences for training, 3,000 for validation, and 5,000 for testing.

4.1.1 Results

Table 4.1 shows the performance of various models on the $10 \rightarrow 10$ task: generating 10 future frames given the 10 previous frames. We use the structural similarity index measure (SSIM) (Bengio et al., 2015) and per-frame mean squared error (MSE) for evaluation. The SSIM ranges between -1 and +1, representing the similarity between the generated image and the ground truth image.

Model	SSIM	MSE
ConvLSTM (Shi et al., 2015)	0.713	96.5
DFN (Brabandere et al., 2016)	0.726	89.0
CDNA (Finn, Goodfellow, and Levine, 2016)	0.728	84.2
FRNN (Oliu, Selva, and Escalera, 2018)	0.819	68.4
VPN Baseline (Kalchbrenner et al., 2016)	0.870	64.1
PredRNN (Wang et al., 2017)	0.869	56.5
PredRNN++ (Wang et al., 2018)	0.885	46.3
E3D-LSTM (Wang et al., 2019)	0.910	41.3
E3D-LSTM Baseline	0.880	69.8
E2D-LSTM with [1,5,5] kernel	0.862	75.0
E3D-LSTM with Residuals	0.890	59.1
E3D-LSTM Finetuned with l_2 only	0.9199	**39.5**
E3D-LSTM Finetuned with Residuals and $l_1 + l_2$	**0.9219**	42.5

TABLE 4.1: Results on the Moving MNIST Dataset for the $10 \rightarrow 10$ task. Higher SSIM or lower MSE scores indicate better results. *Top:* Previous state-of-the-art models. *Bottom:* Our experiments finetuned from pretrained weights demonstrating effectiveness of residual connections.

The top section reports previously evaluated state-of-the-art models in the literature. The middle section describe our baselines obtained by training from scratch.

We make two important observations: (a) using 2D-Conv instead of 3D-Conv inside the recurrent cell results in a SSIM drop of 0.018; (b) residual connections improve the SSIM performance by 0.01. The bottom section shows the effectiveness of fine-tuning with added residual connections, highlighted by the entries in bold.

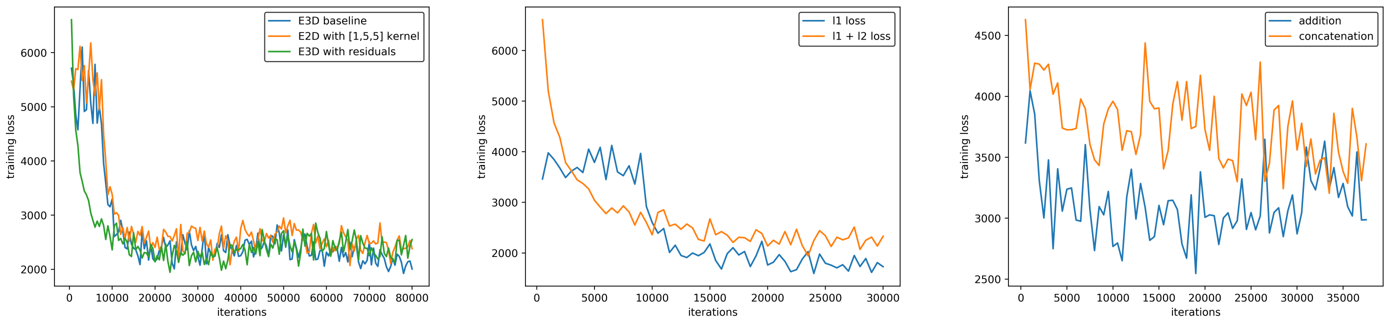


FIGURE 4.1: *Left*: Comparison of E3D baseline, E2D, and E3D with residual connections. Residuals slightly improve training efficiency and result in faster convergence. *Middle*: E3D with residual connections comparing equally weighted l_1 + l_2 loss vs. only l_1 loss. l_1 loss alone improves final training loss. *Right*: Comparing addition and concatenation operators for residual connections.

Figure 4.1 (Left) demonstrates that residuals (green) slightly improve the training efficiency and results in faster convergence. Figure 4.1 (Middle) suggests that l_1 dominates the training process after 10k iterations. Figure 4.1 (Right) shows our findings on choosing to use the addition operation for residual connections as opposed to concatenation.

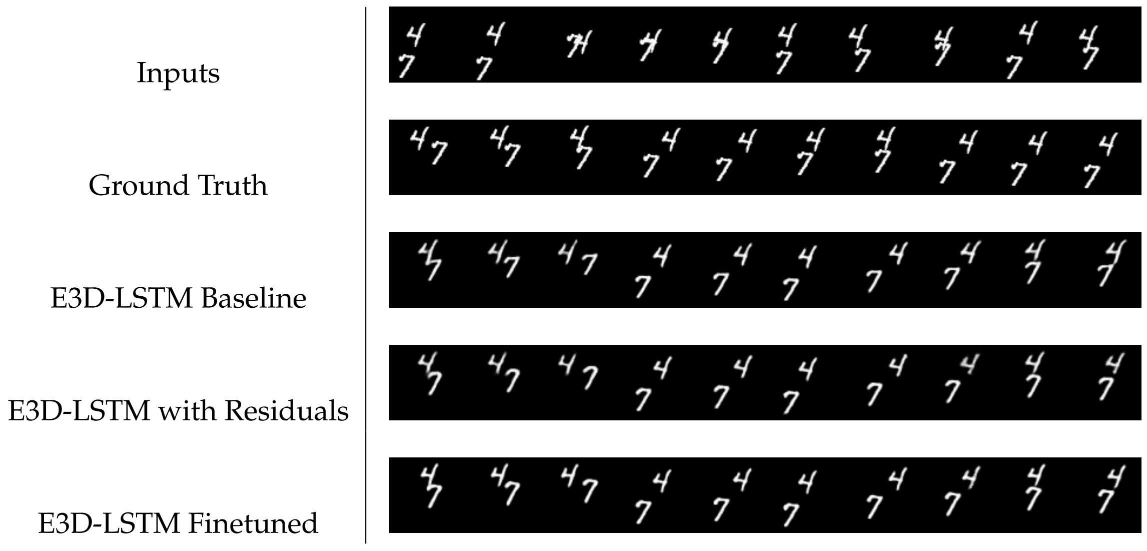


FIGURE 4.2: Video prediction examples from Moving MNIST dataset on the 10 $\rightarrow$ 10 task with '4' and '7'. E3D-LSTM Finetuned with Residuals and $l_1 + l_2$ loss demonstrates slightly improved qualitative image clarity.

4.2 Experiment 2: KTH Action Dataset

Dataset. The KTH Action Dataset (Schüldt, Laptev, and Caputo, 2004) contains 25 individuals performing 6 types of actions including walking, jogging, running, boxing, hand waving, and hand clapping. On average, each video clip lasts for 4 seconds. We split persons 1-16 for training and persons 17-25 for testing. Each frame is resized to 128 × 128 pixels.

4.2.1 Results

Table 4.2 shows the quantitative results on the KTH Action Dataset. We use SSIM and peak signal-to-noise ratio (PSNR) as metrics. Consistent with the observations from the Moving MNIST experiments, we find that the finetuned E3D-LSTM model with residual connections slightly improve performance.

Model	PSNR	SSIM
ConvLSTM (Shi et al., 2015)	23.58	0.712
DFN (Brabandere et al., 2016)	27.26	0.794
FRNN (Oliu, Selva, and Escalera, 2018)	26.12	0.771
PredRNN (Wang et al., 2017)	27.55	0.839
PredRNN++ (Wang et al., 2018)	28.47	0.865
E3D-LSTM (Wang et al., 2019)	29.31	0.879
E3D-LSTM Baseline	27.73	0.854
E2D-LSTM with [1,5,5] kernel	23.30	0.838
E3D-LSTM with Residuals	27.61	0.863
E3D-LSTM Finetuned with Residuals and $l_1 + l_2$	**29.67**	**0.881**

TABLE 4.2: Results on the KTH Action dataset for the 10 → 20 task. Higher PSNR and SSIM scores indicate better performance. *Top:* Previous state-of-the-art models and results. *Middle:* Our experiments trained from scratch. *Bottom:* Our experiments finetuned from pretrained weights demonstrating effectiveness of residual connections.

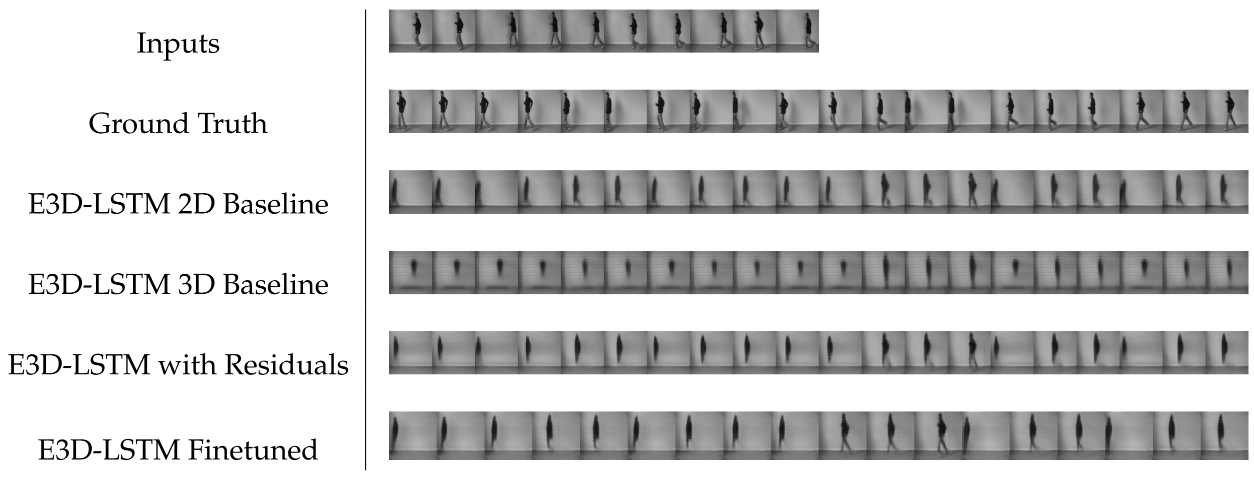


FIGURE 4.3: Video prediction examples from KTH Action dataset on the 10 → 20 task. E3D-LSTM Finetuned with Residuals and $l_1 + l_2$ loss demonstrates slightly improved qualitative image clarity.

4.3 Discussion

We make three observations:

(i) 3D-Conv inside the recurrent cell is important for modeling spatiotemporal data for long term video prediction. Similar to Santoro et al., 2018, this suggests that 2D hidden state transitions are insufficient in capturing both short-term motions and long-term dependencies in spatiotemporal data.

(ii) Residual connections in stacked-LSTM architectures exhibit small empirical improvements due to their ability to maintain crucial spatial information from earlier memory layers, consistent with recent advancements in neural machine translation (Wu et al., 2016).

(iii) Balance between l_1 and l_2 norm components of the loss are crucial in maintaining a stable training process. l_1 incentivizes a sparse solution, while l_2 gives a solution with small magnitudes. Balancing them means balancing a trade-off between sparsity and magnitude of non-zero entries that results in better generalization.

Conclusion

5.1 Summary

In this work, we have implemented the Eidetic 3D LSTM with deep residual connections, building off previous works inspired by analog and spatial representations of mental imagery. We have examined the behavior of 2D-Conv vs. 3D-Conv inside the RNN cell, a variety of residual connection choices, and a class of loss functions consisting of a balance between l_1 and l_2 norm components. We evaluate our approach on the task of spatiotemporal video prediction on two benchmark datasets: Moving MNIST and KTH Action. Our experimental results on baseline models trained from scratch demonstrate that further hyperparameter tuning may be necessary for achieving state-of-the-art performance (0.880 vs. 0.910 SSIM on Moving MNIST; 0.854 vs. 0.879 SSIM on KTH Action). Additionally, our results from finetuned models suggest that using residual connections and a balance between l_1 and l_2 loss allows for more efficient training with faster convergence as well as slightly improved final performance. Our open source implementation is available at `https://github.com/kevinstan/video_prediction`.

5.2 Future Work

Due to time and computational resource constraints, hyperparameter tuning has been kept to a minimum. Were computational resources more plentiful, further work could investigate using neural architecture search (NAS) with reinforcement learning (Zoph and Le, 2017) as a way to enhance spatiotemporal predictive models such as E3D-LSTM. NAS alleviates the difficultly of hand-designing neural network architectures and hyperparameters by using a RNN to generate model descriptions of neural network architectures. In our case, further work could allow for hand-crafted design choices (e.g. the number of stacked layers, the placement of residual connections and their respective operations, the kernel filter shape inside convolution operations in both the RNN cell and the decoder) to be learned automatically.

Another option for accelerating training efficiency and faster model convergence would be to leverage neuro-symbolic concept learning (Mao et al., 2019) inspired by more traditional, proprositional theories of mental imagery. Neuro-symbolic concept learning learns visual concepts without explicit supervision, and builds an object-based scene representation as well as a set of executable, symbolic programs. Similar to how propositional representations are generated through natural language, symbolic learners learn visual concepts based on the language description of the object being referred to. This line of work could potentially shed some light on the debate between analog and propositional theories as well as simultaneously advance the state-of-the-art empirical performance of video prediction models.

Bibliography

Abadi, Martín et al. (2016). “TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems”. In: *CoRR* abs/1603.04467. arXiv: `1603.04467`. URL: `http://arxiv.org/abs/1603.04467`.

Bengio, Samy et al. (2015). “Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks”. In: *CoRR* abs/1506.03099. arXiv: `1506.03099`. URL: `http://arxiv.org/abs/1506.03099`.

Brabandere, Bert De et al. (2016). “Dynamic Filter Networks”. In: *CoRR* abs/1605.09673. arXiv: `1605.09673`. URL: `http://arxiv.org/abs/1605.09673`.

Finn, Chelsea, Ian J. Goodfellow, and Sergey Levine (2016). “Unsupervised Learning for Physical Interaction through Video Prediction”. In: *CoRR* abs/1605.07157. arXiv: `1605.07157`. URL: `http://arxiv.org/abs/1605.07157`.

Graves, Alex (2013). “Generating Sequences With Recurrent Neural Networks”. In: *CoRR* abs/1308.0850. arXiv: `1308.0850`. URL: `http://arxiv.org/abs/1308.0850`.

He, Kaiming et al. (2015). “Deep Residual Learning for Image Recognition”. In: *CoRR* abs/1512.03385. arXiv: `1512.03385`. URL: `http://arxiv.org/abs/1512.03385`.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). “Long Short-Term Memory”. In: *Neural Comput.* 9.8, pp. 1735–1780. ISSN: 0899-7667. DOI: `10.1162/neco.1997.9.8.1735`. URL: `http://dx.doi.org/10.1162/neco.1997.9.8.1735`.

Hornik, Kurt (1991). “Approximation Capabilities of Multilayer Feedforward Networks”. In: *Neural Netw.* 4.2, pp. 251–257. ISSN: 0893-6080. DOI: `10.1016/0893-6080(91)90009-T`. URL: `http://dx.doi.org/10.1016/0893-6080(91)90009-T`.

Huang, Gao, Zhuang Liu, and Kilian Q. Weinberger (2016). “Densely Connected Convolutional Networks”. In: *CoRR* abs/1608.06993. arXiv: `1608.06993`. URL: `http://arxiv.org/abs/1608.06993`.

Ioffe, Sergey and Christian Szegedy (2015). “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *CoRR* abs/1502.03167. arXiv: `1502.03167`. URL: `http://arxiv.org/abs/1502.03167`.

Kalchbrenner, Nal et al. (2016). “Video Pixel Networks”. In: *CoRR* abs/1610.00527. arXiv: `1610.00527`. URL: `http://arxiv.org/abs/1610.00527`.

Kingma, Diederik P. and Jimmy Ba (2014). *Adam: A Method for Stochastic Optimization*. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015. URL: `http://arxiv.org/abs/1412.6980`.

Kosslyn, Stephen M. (1994). *Image and Brain: The Resolution of the Imagery Debate*. Cambridge, MA, USA: MIT Press. ISBN: 0-262-11184-5.

Lecun, Yann et al. (1998). “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE*, pp. 2278–2324.

Mao, Jiayuan et al. (2019). “The Neuro-Symbolic Concept Learner: Interpreting Scenes, Words, and Sentences From Natural Supervision”. In: *International Conference on Learning Representations*. URL: `https://openreview.net/forum?id=rJgMlhRctm`.

McCulloch, Warren S. and Walter Pitts (1943). "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4, pp. 115–133. ISSN: 1522-9602. DOI: 10.1007/BF02478259. URL: https://doi.org/10.1007/BF02478259.

Oliu, Marc, Javier Selva, and Sergio Escalera (2018). "Folded Recurrent Neural Networks for Future Video Prediction". In: *The European Conference on Computer Vision (ECCV)*.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1986). "Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1". In: ed. by David E. Rumelhart, James L. McClelland, and CORPORATE PDP Research Group. Cambridge, MA, USA: MIT Press. Chap. Learning Internal Representations by Error Propagation, pp. 318–362. ISBN: 0-262-68053-X. URL: http://dl.acm.org/citation.cfm?id=104279.104293.

Santoro, Adam et al. (2018). "Relational recurrent neural networks". In: *CoRR* abs/1806.01822. arXiv: 1806.01822. URL: http://arxiv.org/abs/1806.01822.

Schüldt, Christian, Ivan Laptev, and Barbara Caputo (2004). "Recognizing human actions: A local SVM approach". In: vol. 3, 32 –36 Vol.3. ISBN: 0-7695-2128-2. DOI: 10.1109/ICPR.2004.1334462.

Shepard, N. and Jacqueline Metzler (1971). "Mental rotation of three-dimensional objects". In: *Science*, pp. 701–703.

Shi, Xingjian et al. (2015). "Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting". In: *CoRR* abs/1506.04214. arXiv: 1506.04214. URL: http://arxiv.org/abs/1506.04214.

Sutskever, Ilya, Oriol Vinyals, and Quoc V Le (2014). "Sequence to Sequence Learning with Neural Networks". In: ed. by Z. Ghahramani et al., pp. 3104–3112. URL: http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf.

Tieleman, T. and G. Hinton (2012). *Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude*. COURSERA: Neural Networks for Machine Learning.

Wang, Yunbo et al. (2017). "PredRNN: Recurrent Neural Networks for Predictive Learning using Spatiotemporal LSTMs". In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon et al. Curran Associates, Inc., pp. 879–888. URL: http://papers.nips.cc/paper/6689-predrnn-recurrent-neural-networks-for-predictive-learning-using-spatiotemporal-lstms.pdf.

Wang, Yunbo et al. (2018). "PredRNN++: Towards A Resolution of the Deep-in-Time Dilemma in Spatiotemporal Predictive Learning". In: *CoRR* abs/1804.06300. arXiv: 1804.06300. URL: http://arxiv.org/abs/1804.06300.

Wang, Yunbo et al. (2019). "Eidetic 3D LSTM: A Model for Video Prediction and Beyond". In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=B1lKS2AqtX.

Wu, Yonghui et al. (2016). "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation". In: *CoRR* abs/1609.08144. URL: http://arxiv.org/abs/1609.08144.

Zeiler, Matthew D. (2012). "ADADELTA: An Adaptive Learning Rate Method". In: *CoRR* abs/1212.5701. arXiv: 1212.5701. URL: http://arxiv.org/abs/1212.5701.

Zoph, Barret and Quoc V. Le (2017). "Neural Architecture Search with Reinforcement Learning". In: URL: https://arxiv.org/abs/1611.01578.