Video Prediction Using Deep Learning in the Wavelet Domain

Ian Jacobsen

***Abstract*— The goal of this project was to design a system which is capable of video frame prediction. Various design frameworks were evaluated, such as deep learning with traditional L_2 loss, deep learning with adversarial training, deep learning with traditional L_2 loss in the wavelet domain, and deep learning with adversarial training in the wavelet domain**

I. INTRODUCTION

A system which is capable of accurate video prediction has applications in many areas. Although short term prediction is trivial for humans, the problem is much more complex for machines. The intuition that humans develop for forecasting short term events is based on years of learning through observation. Because there are no generalized rules for video prediction, it is not possible to explicitly specify a set of dynamical equations that all videos will follow. Our goal is to train multiple networks with varying training schemes on a dataset that follows a well defined set of dynamics, and evaluate the performances of each network.

II. OVERVIEW

A. Problem Formation

1) Traditional L_2 Loss: The most simple network that was designed in this experiment consisted of a convolutional LSTM [4] which was trained to minimize the L_2 loss, defined as:

$$L_{L_2}(V_{gen}) = \sum_{x,y,t} (V_{truth}(x,y,t) - V_{gen}(x,y,t))^2 \quad (1)$$

Minimizing the L_2 loss corresponds to increasing the accuracy in the sequence prediction [3].

2) Adversarial Training: The adversarial training scheme [2] required two networks; *G* and *D*. *G* was selected to be a convolutional LSTM, and *D* was selected to be a 3D CNN. The reasoning for selecting a convolutional LSTM for the architecture of *G* is because it is well suited for dealing with spatiotemporal data [4] [5] *D* is simply used for binary classification of sequences, and therefore a 3D CNN was chosen to be the architecture. The 3D CNN was selected because the input data is a stack of 2D frames, thus the data is three dimensional.

The loss function for *D* is simply the cross-entropy between the true and predicted labels. More specifically, we define:

$$D(V) = -\sum_i (y_i \log \hat{y_i} + (1-y_i) \log 1 - \hat{y_i}) \quad (2)$$

In the adversarial setting, the loss function for *G* consists of two terms; a term for the L_2 loss, and a term for the adversarial loss. The L_2 term corresponds to accuracy in sequence prediction, and the adversarial term corresponds to the ability of the generated sequence's temporal motion to fool *D*.

$$L_{gen}(V_{gen}) = L_{L_2}(V_{gen}) + \lambda L_{adv}(V_{gen}) \quad (3)$$

More specifically, we have:

$$L_{gen}(V_{gen}) = \sum_{x,y,t} (V_{truth}(x,y,t) - V_{gen}(x,y,t))^2 + \lambda \log(1 - D(V_{gen})) \quad (4)$$

where $D(V_{gen})$ is the output of *D* when a generated sequence is the input.

B. Data

The data that was used in this experiment was a synthetic set of bouncing balls. The reason why this dataset was chosen was because it has clear dynamics which we believed networks would be capable of learning a representation for. It is important that the networks are able to learn some understanding of the data because the performance of each network is to be compared.

C. Tasks

1) Design of Convolutional LSTM in TensorFlow [1].
2) Design of 3D CNN in TensorFlow [1].
3) Implementation of Traditional L_2 Loss Training.
4) Implementation of Adversarial Training.
5) Implementation of Wavelet Decomposition.
6) HPC Batch Scripts.
7) Evaluation of Models and Data Processing.

III. PROJECT ACCOMPLISHMENT

1) Designing the Convolutional LSTM: The convolutional LSTM was defined to have two convolutional encoding layers at the input of the LSTM cell, and two decoding layers at the output of the LSTM cell.

The first encoding layer has a kernel size of 3×3, a stride of 2, and outputs 9 feature maps. The second encoding layer has a kernel size of 3×3, a stride of 1, and outputs 9 feature maps. The convolutional LSTM cell uses a kernel size of 3×3, a stride of 1, outputs 9 feature maps, and passes a hidden state. The first decoding layer has a kernel size of

.Department of Electrical Engineering, New York University, New York, New York. ij405 at nyu.edu

3×3, a stride of 1, and outputs 9 feature maps. The final decoding layer has a kernel size of 3×3, stride of 2, and outputs 1 feature map.

The convolutional LSTM begins outputting predictions of the next frame after the input sequence has reached a specified length. In our experiment, the total sequence length was 10 frames, and the convolutional LSTM would begin making predictions after frame number 6. The reason for the network not outputting predictions for the first 6 frames is that it needs time to infer the hidden state. Without a good estimate of the hidden state, the estimated output frames will not be accurate because the network does not understand in which direction the balls are moving in until it observes a short sequence of frames.

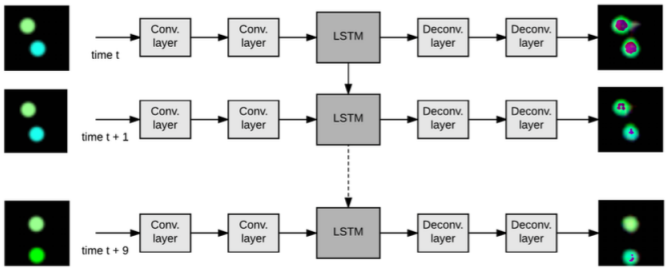

Fig. 1. Visualization of Convolutional LSTM

2) Designing the 3D CNN: The 3D CNN was given a very simple design. There is only one convolutional layer, one fully connected layer, and one output layer in the network. The kernel size for the convolutional layer is $3 \times 3 \times 3$, with a stride of 1, and an output of 9 feature maps. Because there is no downsampling in this architecture, the fully connected layer has $32 \times 32 \times 9 = 9216$ inputs to 81 hidden units, and the output layer has 81 inputs to 2 hidden units.

3) Adversarial Training Implementation: An adversarial training scheme was used to train the two networks. In order to train the two networks in tandem, a GAN class was created in Python. The GAN class defined methods for both networks. The essential methods are for network optimization, network prediction, and network loss.

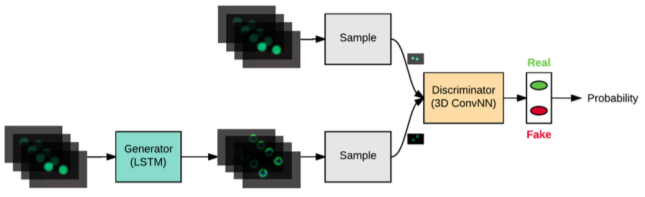

Fig. 2. Visualization of GAN Framework

4) Wavelet Transform Implementation: A one level Haar transform was implemented to decompose the data sequences into wavelet subbands. The forward transform was implemented with analysis filter banks, and the inverse transform was implemented with synthesis filter banks.

5) Running on HPC: Batch scripts were written to train the networks on the GPUs available on the HPC cluster Prince. The scripts requested 20GB of memory and 40 hours of runtime on two GPUs.

6) Data Analysis: Python scripts were written to parse the output files to perform data analysis on the networks. Important plots such as loss v.s. iterations provide insight as to how well the networks will do during testing.

IV. Results

A. Traditional L_2 Training

As we can see from 3, the L_2 loss trend decreases with the number of iterations. This is ideal.

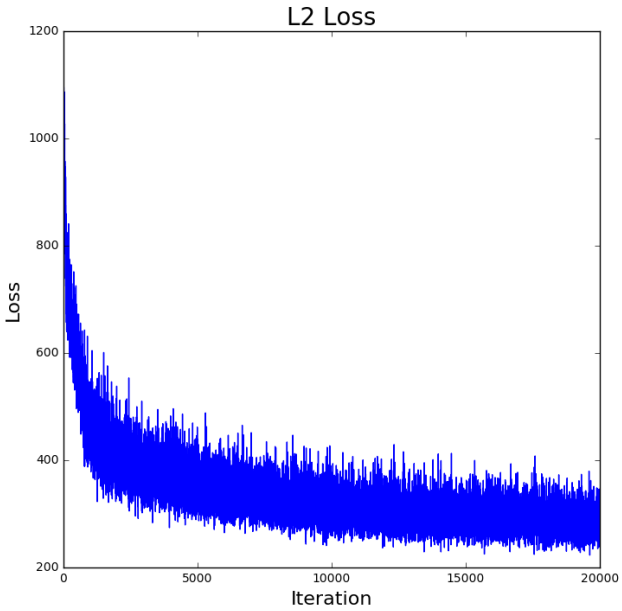

Fig. 3. Loss v.s. Iteration Number of Traditional L_2 Training in Data Domain

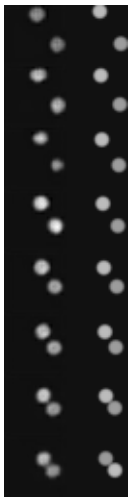

Fig. 4. Predicted and True Frames with L_2 Loss Network

The left hand side column of 4 shows a sequence of predicted frames using the L_2 loss model, and the corresponding true frames are on the right hand side.

B. Adversarial Training

As we can see from 5, the total loss trend decreases with the number of iterations. This is ideal.

The green curve shows the adversarial loss, which goes to zero. This means that the generator has learned how to generate sequences which fool D. From this result we can see that the GAN never reached equilibrium, which is 50% classification accuracy from D. Although the network did not reach the equilibrium state, the adversarial training was still useful in training G. We can say this because by inspection of 3 and 5, it is clear that the total loss in 5 decreases quicker than the loss in 3.

One way to overcome the fact that G is able to fool D is by changing the architecture of D. By making D more complex, D may become better at classifying sequences. This would be useful in training G to produce more realistic sequences.

The blue curve represents the L_2 loss, and the red curve represents the total loss (i.e. the sum of the L_2 loss and adversarial loss).

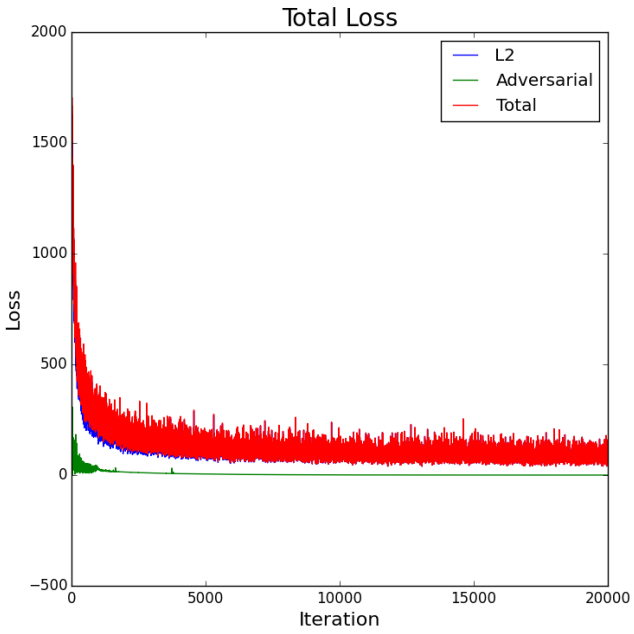

Fig. 5. Loss v.s. Iteration Number of Adversarial Training in Data Domain

The left hand side column of 6 shows a sequence of predicted frames using the adversarial model, and the corresponding true frames are on the right hand side.

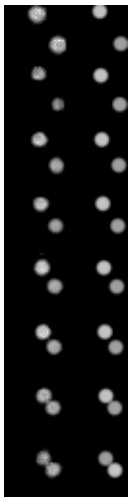

Fig. 6. Predicted and True Frames with Adversarial Network

C. Traditional L_2 Training in the Wavelet Domain

As we can see from 7, the L_2 loss trend decreases with the number of iterations. This is ideal.

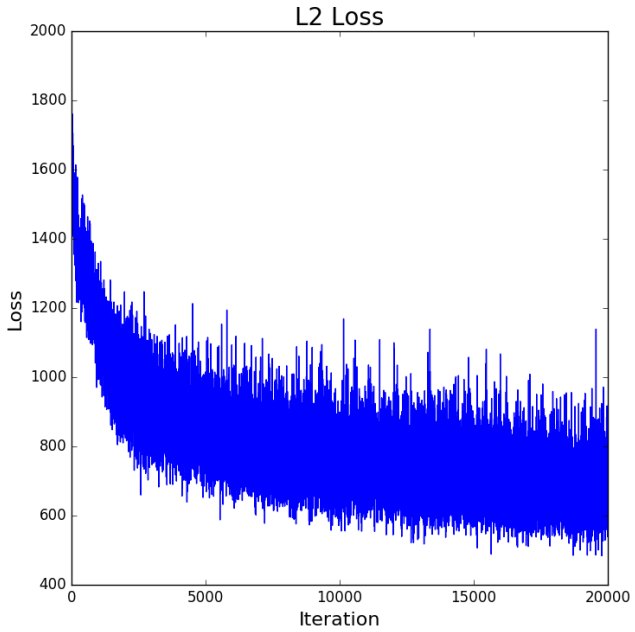

Fig. 7. Loss v.s. Iteration Number of Traditional L_2 Training in Coarse Subband

As we can see from 8, the L_2 loss trend decreases with the number of iterations. This is ideal.

The left hand side column of 9 shows a sequence of predicted frames using the L_2 loss model in the wavelet domain, and the corresponding true frames are on the right hand side. The frames were predicted independently in each individual subband, then the frames were sent through a

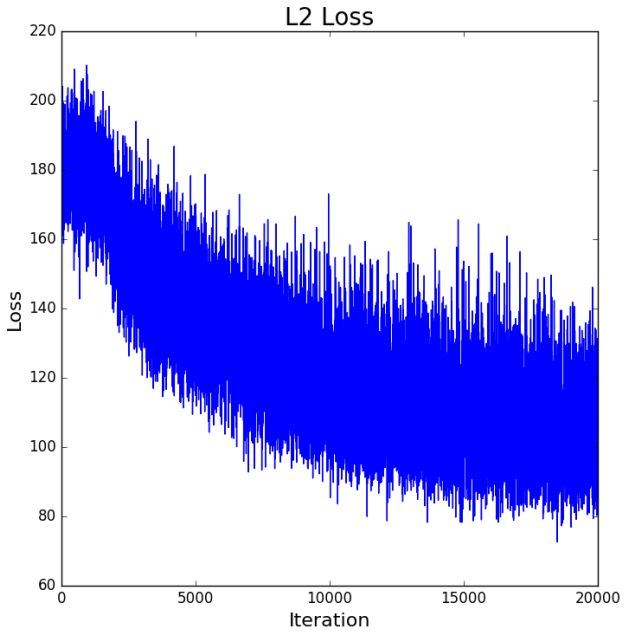

Fig. 8. Loss v.s. Iteration Number of Traditional L_2 Training in Detail Subband

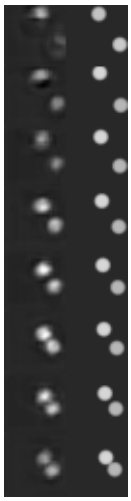

Fig. 9. Predicted and True Frames with L_2 Loss Network in the Wavelet Domain

synthesis filter bank to reconstruct the final prediction.

The left hand side column of 10 shows a sequence of predicted detail subband frames using the L_2 loss model, and the corresponding true detail subband frames are on the right hand side.

The left hand side column of 10 shows a sequence of predicted coarse subband frames using the L_2 loss model, and the corresponding true coarse subband frames are on the right hand side.

D. Adversarial Training in the Wavelet Domain

As we can see from 12, the network does not converge. The loss v.s. iteration curve is interesting. Both the L_2 loss and the adversarial loss start off decreasing, but near iteration 5000 the training stops behaving the way we would hope. The L_2 loss becomes approximately constant for the remainder of training, and the adversarial loss goes to zero.

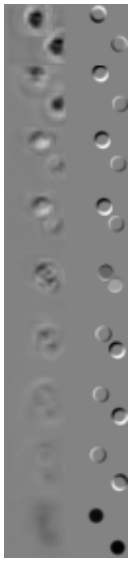

Fig. 10. Predicted and True Detail Subband Frames with L_2 Loss Network

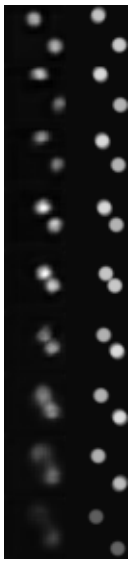

Fig. 11. Predicted and True Coarse Subband Frames with L_2 Loss Network

As we can see from 13, the network does not converge. The total loss remains approximately constant throughout training. This is not ideal. The large spikes in the total loss are due to the adversarial loss. Where these spikes occur, D is able to correctly classify the generated sequences as coming from G.

An explanation for the adversarial framework not training correctly on the subband data is because the λ parameter is too large and the architecture of D is not strong enough to accurately pick up on the differences between true and generated sequences. Thus, the adversarial loss takes control and the generator learns to fool D by outputting sequences

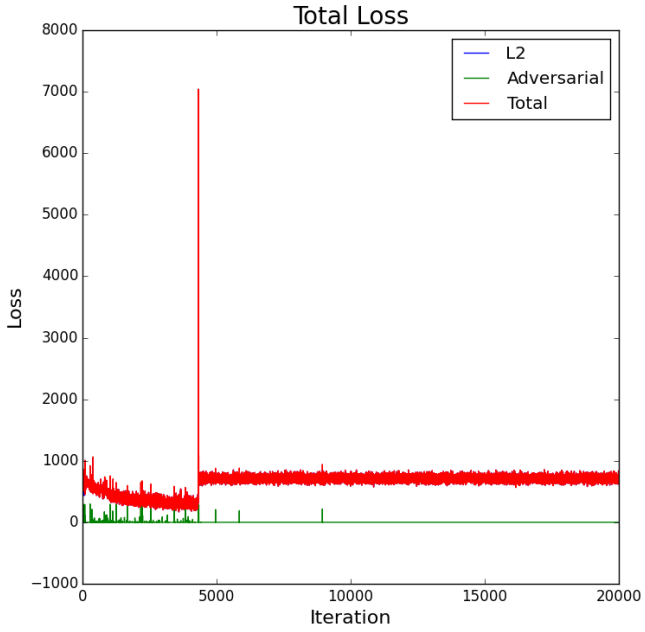

Fig. 12. Loss v.s. Iteration Number of Adversarial Training in Coarse Subband

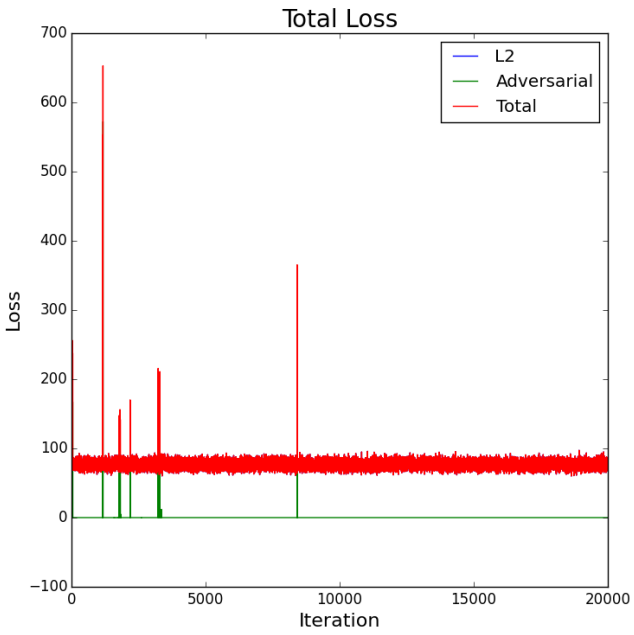

Fig. 13. Loss v.s. Iteration Number of Adversarial Training in Detail Subband

which the weak D classifies as true. In this experiment, the output frames are uniformly black. This means that D cannot differentiate between an all black sequence and a true sequence, and therefore since the adversarial loss for G is zero, the total loss of G is only the L_2 loss between a sequence of all black frames and the true frames. To address this problem, we may increase the complexity of D, and lower the value of λ.

V. Summary

In the setting of this experiment it is clear that using deep learning to perform video prediction in the wavelet domain does not provide an advantage over viHdeo prediction in the data domain. Further investigation with different wavelet transforms may lead to a different conclusion, but for the one level Haar transform we can see that the predicted video is of worse quality than the video predicted directly from the data domain. Furthermore, we can see that performing video prediction in the data domain using a model that has been trained via an adversarial process gives slightly better video predictions than just using the convolutional LSTM on it's own. For future work, it may be interesting to compare the performance of the GAN network in the wavelet domain with a lower value for λ and a more complex D architecture than what was initially used in this experiment.

APPENDIX

References

[1] 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, Dan 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. Software available from tensorflow.org.

[2] Ian J. Goodfellow. NIPS 2016 tutorial: Generative adversarial networks. *CoRR*, abs/1701.00160, 2017.

[3] Michaël Mathieu, Camille Couprie, and Yann LeCun. Deep multi-scale video prediction beyond mean square error. *CoRR*, abs/1511.05440, 2015.

[4] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. *CoRR*, abs/1506.04214, 2015.

[5] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhutdinov. Unsupervised learning of video representations using lstms. *CoRR*, abs/1502.04681, 2015.