Video Prediction

Independent Work Report (MAE 340, Spring 2021)
Advisor: Professor Olga Russakovsky

Matthew Coleman

April 26, 2022

This project represents my own work,
in accordance with the University regulations.
/s/Matthew Coleman

Contents

1 Introduction **3**

2 The Task of Video Prediction **3**

3 Families of Prediction Models **4**
- 3.1 Recurrent Neural Networks . . . 4
- 3.2 Long Short-Term Memory (LSTM) Cell . . . 5
- 3.3 Sequence To Sequence Learning . . . 7

4 Experiments **8**
- 4.1 Sequence Prediction . . . 9
  - 4.1.1 Generated Sinusoids . . . 10
  - 4.1.2 Generated Noise . . . 10
  - 4.1.3 Stocks . . . 12
- 4.2 Video Prediction . . . 13
  - 4.2.1 Moving MNIST . . . 14
  - 4.2.2 KTH . . . 16
  - 4.2.3 BAIR . . . 18

5 Conclusion **19**

References **20**

1 Introduction

While humans cannot perfectly predict the future, they are indeed capable of inferring a great deal of information about near events in the future, and this knowledge greatly aids them in planning out their actions, such as which movements to take to reach a goal. This ability to forecast the future is a direct result of an understanding of causality that is learned through observation and interaction [2].

A great amount of human predictions are erroneous in major respects, but even the humans least adept at inferring far-off outcomes and consequences still are masters of learning very near-term ones. For example, humans have a good sense for where a car will move in the street, or which direction a pedestrian may continue walking. Even a young child can predict where to toss a football to a moving receiver, and even this small knowledge reveals an infinite wisdom compared to the most advanced video prediction methods.

The task of video prediction is comprised of several open challenges in computer vision; it uses some of the most recent model architectures that have been developed and it even contends directly with an impossible task altogether, which is to predict the future. Although it is a particularly confusing task, it also has the potential for immense impact and immediate practical applications, such as in autonomous driving [4], video interpolation [6] and most interesting in the context of this report, robotic control systems [3].

This project will examine the current state-of-the-art in video prediction machine learning models, report on several experiments carried out by implementing and testing such a model on various existing datasets, and attempt to make meaningful conclusions about video prediction and learning causality.

2 The Task of Video Prediction

The task of video prediction is to construct an approximation for the completion of a sequence of frames, given only the initial sequence of frames. Formally, given an ordered set of n image frames $\boldsymbol{X} = (X_1, X_2, X_3 \cdots X_n)$, the task is to predict the latter m frames of the sequence $\boldsymbol{Y} = (Y_1, Y_2, Y_3 \cdots Y_m)$, each frame of which having the same dimensions, for example with c channels, height h, and width w. At each inference in training, the model predictions $\hat{\boldsymbol{Y}} = \left(\hat{Y}_1, \hat{Y}_2, \hat{Y}_3 \cdots \hat{Y}_n\right)$, with the same dimensions as the inputs, are conditioned on the input sequence $\boldsymbol{X}$, and the model weights are updated typically by the gradient of a loss function computed between the predictions and ground truth sequence $\boldsymbol{Y}$ directly. Critically, since there is no human intervention or labeling required for the model to do this, and models typically are able to learn from the implicit temporal organiziation of the video data, video prediction is a self-supervised task [7].

3 Families of Prediction Models

Modern video prediction models tend to adopt several canonical architectures, which are normally simple and easily generalizable for specific tasks, such that they can be used as building blocks or blueprints within larger architectures. Understanding what each architecture seeks to do on a high level is imperative to understanding what kind of output one should expect from the model, since they are each very unique, and research implementations make use of them in different ways.

For example, RNNs and generative networks are each successful and well-tested paradigms used in many other machine learning domains outside of computer vision, but they are especially utilized within video prediction because of their key properties and strengths, particularly of RNNs to work on time-sequential data and of generative networks to "imagine" new data within a distribution. These paradigms are used extensively in Convolutional LSTM, FutureGAN [1], and SAVP models [5], as well as many other models in cutting-edge research. A short description of each family and its relevance to video prediction is given below:

3.1 Recurrent Neural Networks

Formally, a Recurrent Neural Network (RNN) is the end-result of a mathematical analysis of a nonlinear first-order non-homogeneous ordinary differential equation describing the evolution of a state signal s as a function of time, along with an input signal x. The canonical statement of an RNN is given below in the form of a discrete Delay Differential Equation (DDE) [9]:

$$\begin{aligned} s_t &= W_s s_{t-1} + W_r r_{t-1} + W_x x_t + \theta_s \\ r_t &= G(s_t) \end{aligned} \tag{1}$$

With W_s, W_r, and W_x as weights which are either multiplied or convolved with their respective signals, and θ_s as a bias term which is typically added in element-wise fashion to s. This equation also includes the read-out or output signal r, which is the activation $G(z)$ of the state signal, and can be seen as the "output" of the network. All together, this equation describes all of the moving parts of an RNN.

As a toy example (and ignoring some technical tricks that would be required to make this actually happen), a network of this type could be capable of transcribing seqeunces of a lecture by evaluating discrete audio samples recorded from the event and outputting the spoken words in text form [9]. Another extremely common application of vanilla RNNs is machine translation, in which the input sequence is text in one language and the output is meant to be the translation of that text into another language.

An easily understandable depiction of the RNN described in Equation 1 can be found in Figure 1. This figure also shows the "unfolding" or "unrolling" of the model, which is just a way of seeing each time-step of the state signal s_t, input signal x_t, and ouput signal r_t (here replaced by $\hat{y}_t$, denoting the network inferences) as they are produced over time.

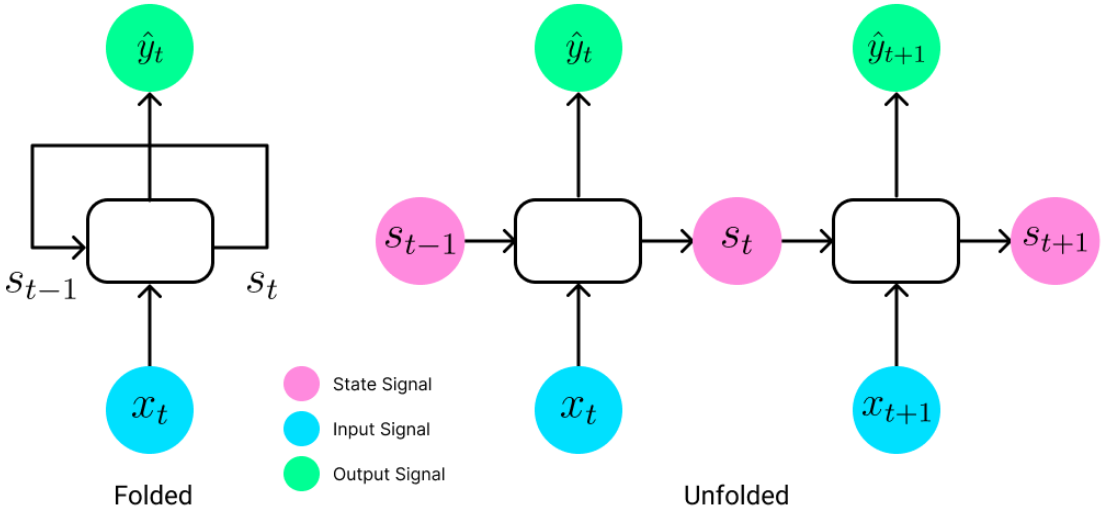

Figure 1: RNN Architecture

RNNs are generally trained using a techinque called Backpropagation Through Time (BPTT), in which the gradient of a loss function taken at a certain instance in time is used to update the model parameters recursively through the history of the model, using the chain rule. In this way, RNN models trained over long sequences have their parameters updated by the product of many Jacobian matrices, which, just like a product of many real numbers, can vanish or explode very easily [8]. For this reason, along with several others having to do with the long-term stability of RNNs, the Long Short-Term Memory (LSTM) cell was developed with nonlinear, data-dependent controls in the form of several "gates" that control input to and output from the cell's state [9].

3.2 Long Short-Term Memory (LSTM) Cell

The changes described in the following section take the form of a modified system of equations [11]:

$$\begin{aligned} f_t &= \sigma(W_{fh}h_{t-1} + W_{fx}x_t + b_f) \\ i_t &= \sigma(W_{ih}h_{t-1} + W_{ix}x_t + b_i) \\ \tilde{c}_t &= \tanh(W_{\tilde{c}h}h_{t-1} + W_{\tilde{c}x}x_t + b_{\tilde{c}}) \\ o_t &= \sigma(W_{oh}h_{t-1} + W_{ox}x_t + b_o) \\ c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \\ h_t &= o_t \circ \tanh(c_t) \end{aligned} \tag{2}$$

The key differences in the LSTM are the four gates denoted by f, i, $\tilde{c}$, and o. Other than these, the cell state c and output h are analogous to the original RNN definition. Now, the input and output gates are able to "learn" how to add information to the cell

state or take information for inferences, and the forget gate is able to remove information from the cell state entirely. To see how this is done, recall that the sigmoid funtion $\sigma(z)$ has an output range of $(0, 1)$, meaning that element-wise multiplication by embeddings activated by sigmoid can minimize and essentially nullify the cell state. In the input gate, this same technique is applied for i onto $\tilde{c}$, and in the output gate it is applied by o onto c, meaning that i decides what from h_{t-1} is allowed into the cell state and o decides what from c_{t-1} is allowed out of the cell state into the inference. A diagram of these connections is shown in Figure 2.

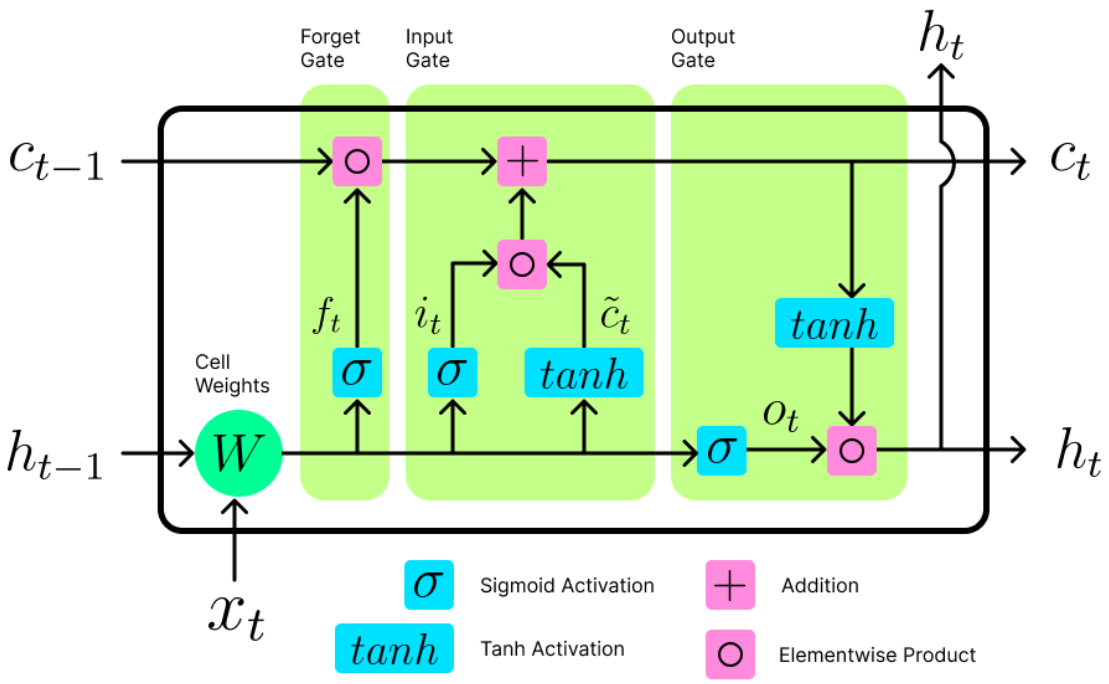

Figure 2: LSTM Cell Architecture

In this way, LSTM cells are capable of mitigating some of the drawbacks of RNNs such as vanishing and exploding gradients, and they are capable of learning how to make good inferences for very different sequences. In other words, they are better able to approximate both y_{t_1} and $y_{t_2 \gg t_1}$ using the same parameters.

Now that some of the general architectures and methodologies are laid out, however, the question still remains of how these models actually predict the continuation $\boldsymbol{Y}$ of a sequence of frames $\boldsymbol{X}$ with any sort of accuracy. In order to do this, one final methodology must be examined, namely Sequence to Sequence (Seq2Seq) learning, which will lead to the final implementation of a Convolutional LSTM used for experimentation in this project.

3.3 Sequence To Sequence Learning

Seq2Seq learning is only a slight modification to the original usage of RNNs, but they represent an elegant solution to a classical shortcoming of most Deep Neural Networks (DNNs), which is that, despite their flexibility, they can only be applied to problems in which inputs and outputs can fit into fixed, discretized, vectors, and therefore must be of a certain length or size. Seq2Seq learning solves this problem for video prediction as well as many other tasks by using 2 LSTMs: an encoder to read the sequence and learn to create an embedding vector, and a decoder, which is passed the embedding vector and learns to generate the predicted sequence [10]. A diagram of this procedure is shown in Figure 3.

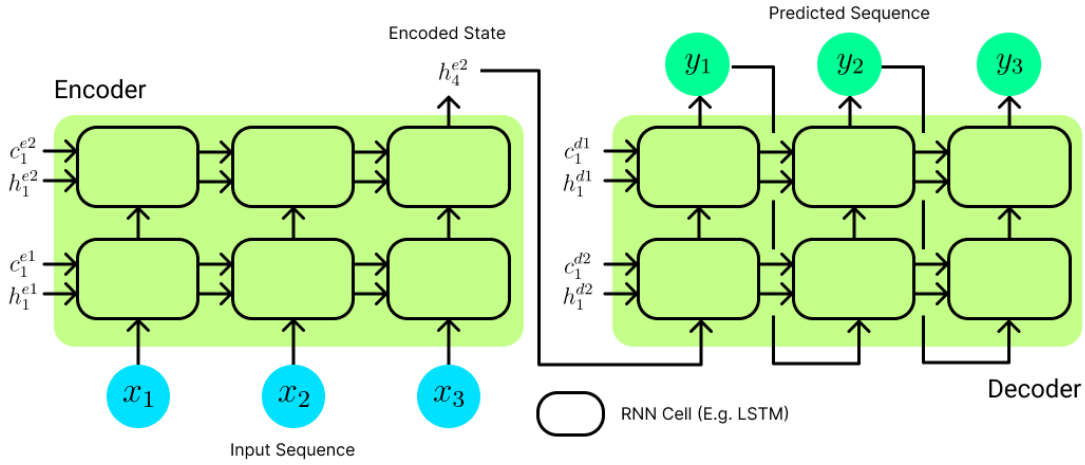

Figure 3: Seq2Seq Architecture

The diagram shows how the input sequence x is fed into the encoder LSTM for 3 steps (in practice this can be any length), how the encoder generates an embedding vector h_4 which is passed to the decoder as an input, and how the decoder generates the predicted sequence by passing the predicted output $\hat{y}_t$ back into itself at the next time step as an input, continuing until all predicted frames are generated.

This diagram also shows two layered LSTM cells, which is another key modification used in this project's implementation. Note that each layer has it's own states h and c, and at each time step, the cells pass their h embedding up to the next cell as an input. Deep, multilayered LSTMs have been shown to significantly outperform shallow LSTMs for language translation [10], however, this project will also analyze the effect of depth on LSTM inferences for video prediction.

In sum, these are the main methods used in this report's implementation of an LSTM model for video prediction, which can be found at this GitHub repository.

4 Experiments

The main experimental procedure carried out in this report is the training and testing of the Convolutional LSTM model on the MovingMNIST, KTH, and BAIR datasets, however, in order to more gain a more robust understanding of LSTM features and limitations, as well as to debug the training mechanisms in a much simpler setting, it was useful to first test a Linear LSTM model on one-dimensional sequential data before moving fully to video prediction. Three datasets of this type were implemented: a dataset consisting of generated sin waves with random frequency and phase offset, a dataset consisting of NASDAQ close price data from 8 tech companies, and a dataset consisting of randomly generated points connected using a cosine interpolation function. The choice of these datasets were intended to each test the model with a varying degree of stochasticity, the sin waves being the most deterministic, or predictable, the random points being the most stochastic, or random, and the stock price data hopefully being somewhere in between.

When researching and implementing a prediction model of any sort, it is important to consider that certain sequences are simply impossible to predict, and that even if the model was "perfect" by human standards, it would still fail to perform this task perfectly. The interesting question here is not whether the model will be able to achieve a certain accuracy in predicting the sequences but rather where or how exactly will it fail? And where it does fail, which features of the model's architecture and methodology are to blame?

All these questions lead directly into one of the major concerns in video prediction research, which is that the task is inherently hard to judge [7], particularly that pixel-wise loss functions, such as the Mean Squared Error (MSE) function used to train this model, cause models to prefer blurry results that average out multiple possibilities for the next time step rather than a single, clearer image that could be very wrong using pixel-wise metrics. Models trained in this way are not directly learning to produce images but merely to appease the loss function, and these results may not be as clear or useful to humans.

This Linear LSTM was implemented with nearly the same exact architecture as the Convolutional LSTM, however with Linear layers in place of convolutions, and as a result, a 2D convolution layer as the final step as opposed to 3D. Each time it was trained on sequences of length 50, for a total duration of 10 epochs.

4.1 Sequence Prediction

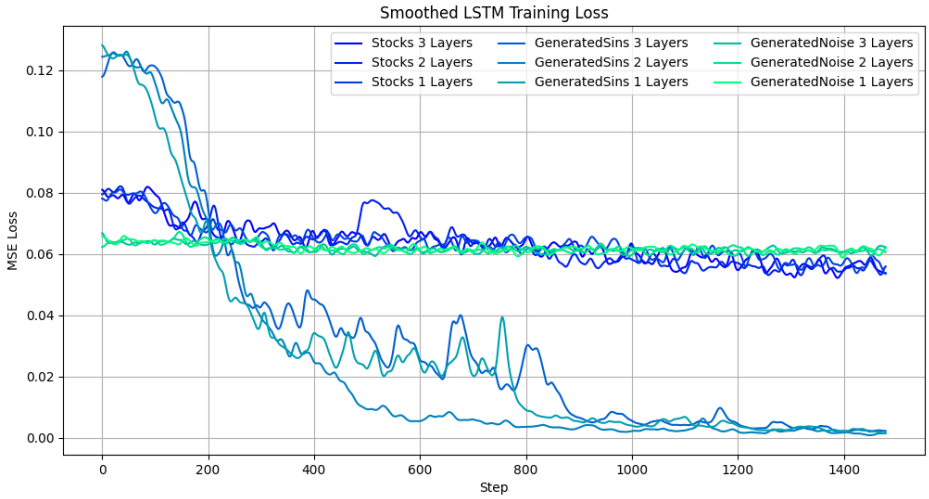

Figure 4: LSTM Training Loss on GeneratedSins, GeneratedNoise, and Stocks Datasets for Models with Varying Number of Cell Layers

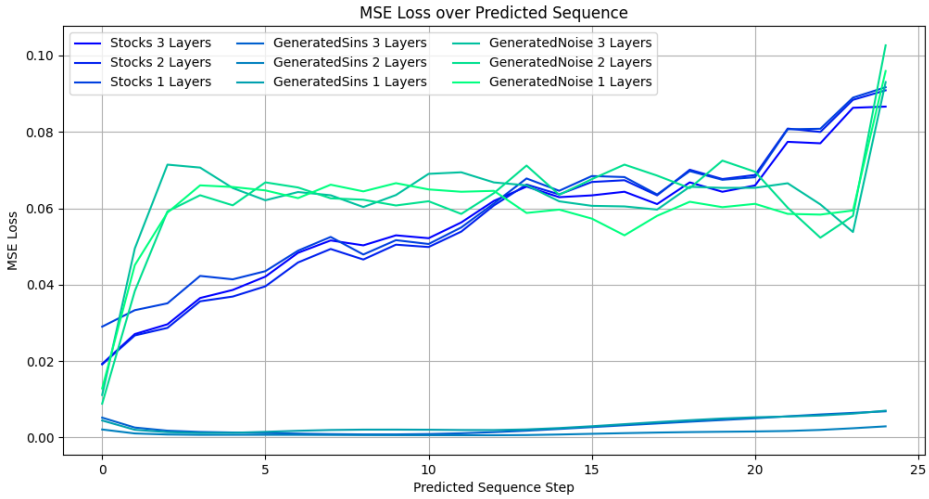

Figure 5: LSTM Test Loss over Predicted Sequence Steps for GeneratedSins, GeneratedNoise, and Stocks Datasets for Models with Varying Number of Cell Layers

4.1.1 Generated Sinusoids

This dataset proved, as it reasonably should, to be the most predictable of the three datasets. This can be observed in the plot of training losses in Figure 5, which shows that GeneratedSins data decreased the most over training out of the three, and in the plot of average test losses over the predicted sequence in Figure 4, which shows that GeneratedSins led to the most accurate predictions over longer sequences by far. Additionally, Figure 6 shows a test inference of a 2-layer LSTM model trained on this dataset with the ground truth sequence underlaid.

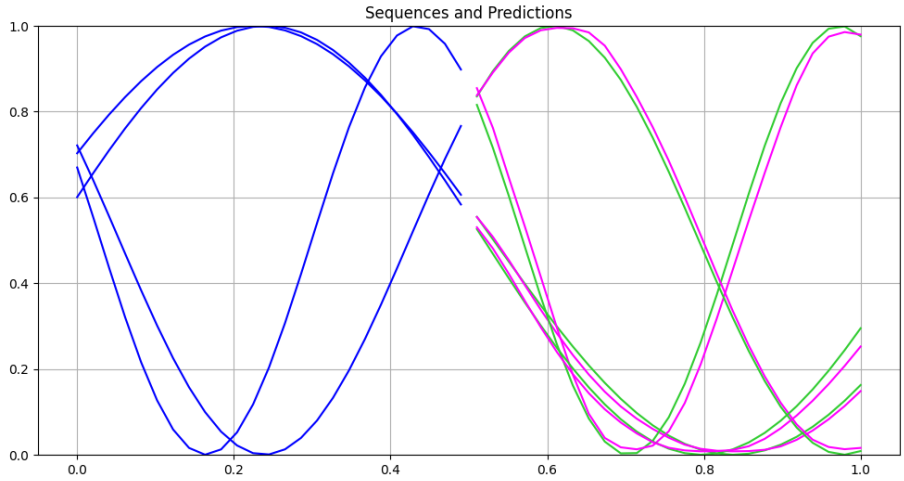

Figure 6: LSTM Inference on GeneratedSins Dataset, with Ground Truth Input X in Blue, Ground Truth Label Y in Green, and Model Prediction in Purple

As clean and impressive as these predictions are, however, they are not perfect, and still decrease in accuracy as the sequence progresses. While the errors are minor, this shortcoming can be blamed directly on the Seq2Seq architecture, which feeds each LSTM cell's output directly back into itself as an input for the next time step in order to generate the full sequence (See Figure 3). This means that if any prediction is off by a tiny amount, the next prediction will likely share and even increase that error. If the error is major such as the FILLIN in Figure **??**, the prediction and ground truth will likely diverge completely, and this would reveal a fundamental "misunderstanding" of the LSTM.

4.1.2 Generated Noise

This dataset, in contrast, was nearly impossible for the model to learn satisfactorily. This model's loss plot in Figure **??** shows that the model is able to follow the random sequences for the first few points until the next random point is interpolated, however

beyond that the model predicts nothing but a straight line centered at $y = 0.5$. While this seems like a failure of some sort, it is in fact a direct demonstration of the model's loss function, Mean Squared Error (MSE), which is a simple function, shown below:

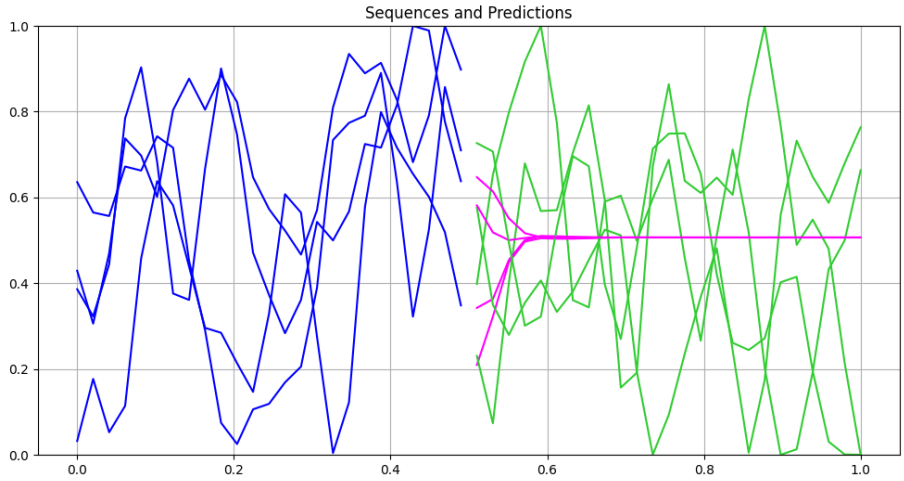

Figure 7: LSTM Inference on GeneratedNoise Dataset, with Ground Truth Input X in Blue, Ground Truth Label Y in Green, and Model Prediction in Purple

$$\text{MSE} = \sum_{i}^{n} \tag{3}$$

4.1.3 Stocks

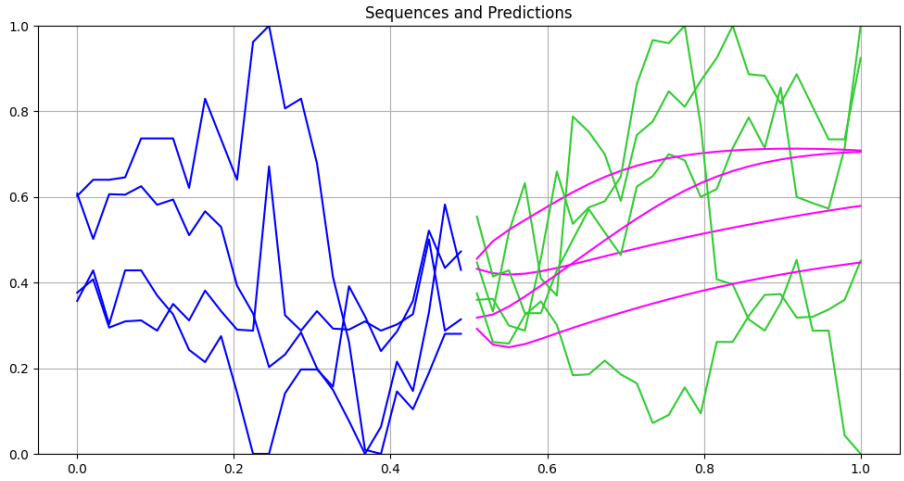

Figure 8: LSTM Inference on Stocks Dataset, with Ground Truth Input X in Blue, Ground Truth Label Y in Green, and Model Prediction in Purple

4.2 Video Prediction

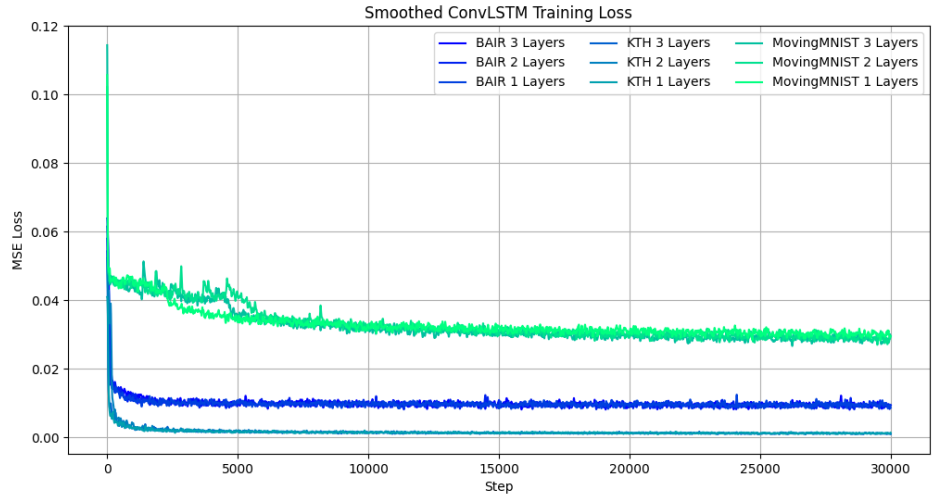

Figure 9: ConvLSTM Training Loss on MovingMNIST, KTH, and BAIR Datasets for Models with Varying Number of Cell Layers

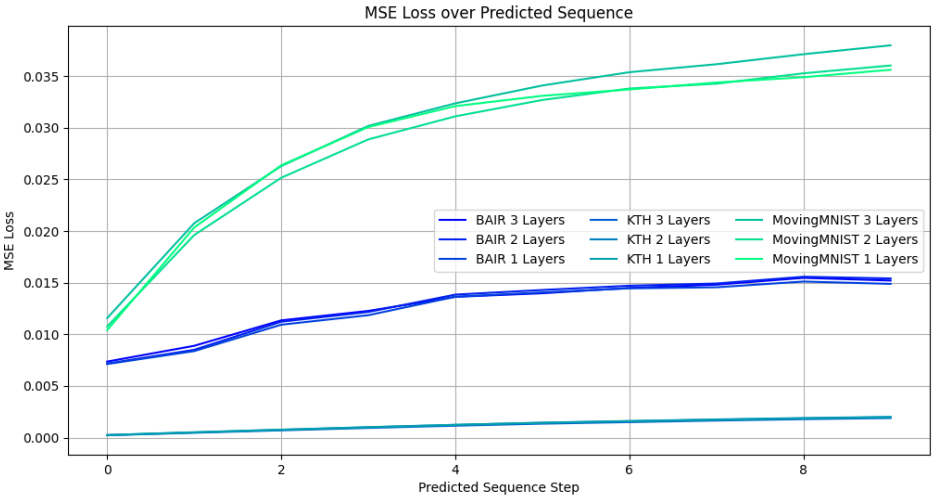

Figure 10: ConvLSTM Test Loss over Predicted Sequence Steps on MovingMNIST, KTH, and BAIR Datasets for Models with Varying Number of Cell Layers

4.2.1 Moving MNIST

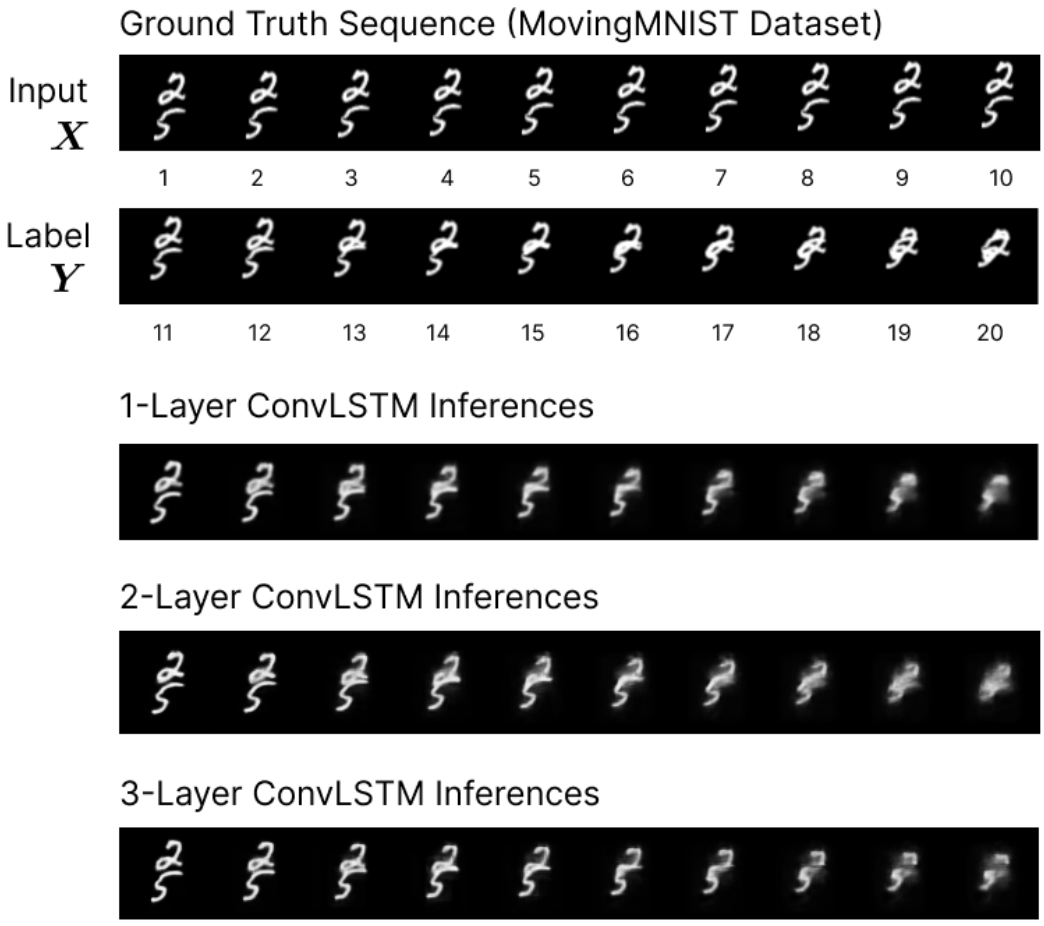

Figure 11: Convolutional LSTM Inference on MovingMNIST dataset

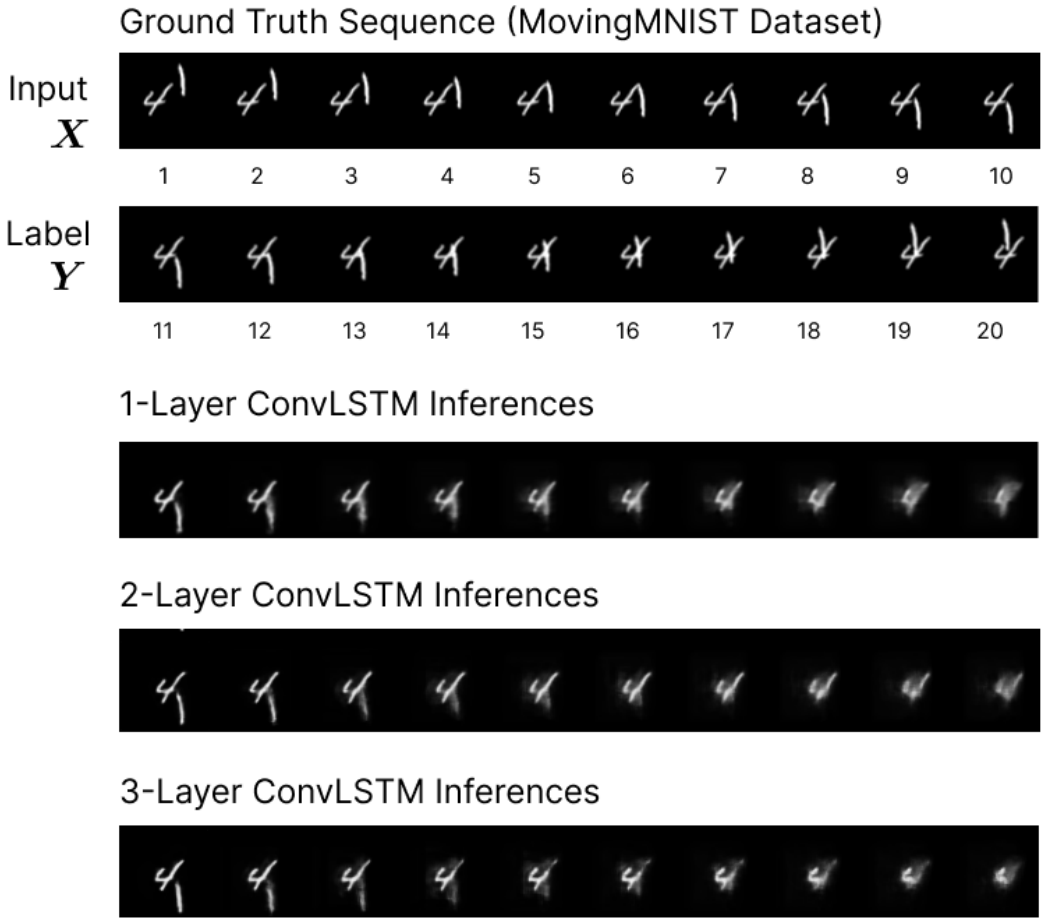

Figure 12: Convolutional LSTM Inference on MovingMNIST dataset

4.2.2 KTH

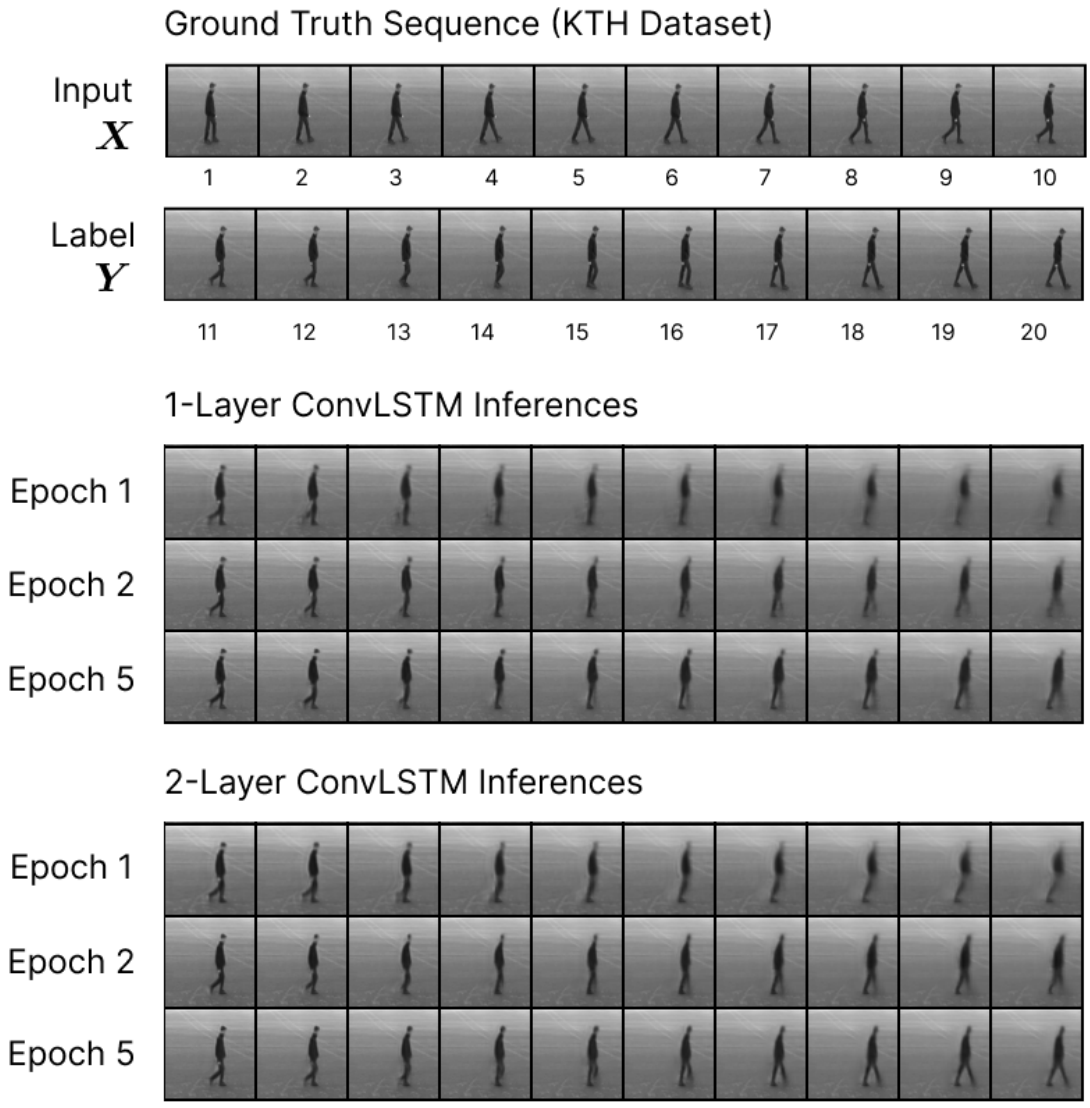

Figure 13: Convolutional LSTM Inference on KTH dataset (Sample 10)

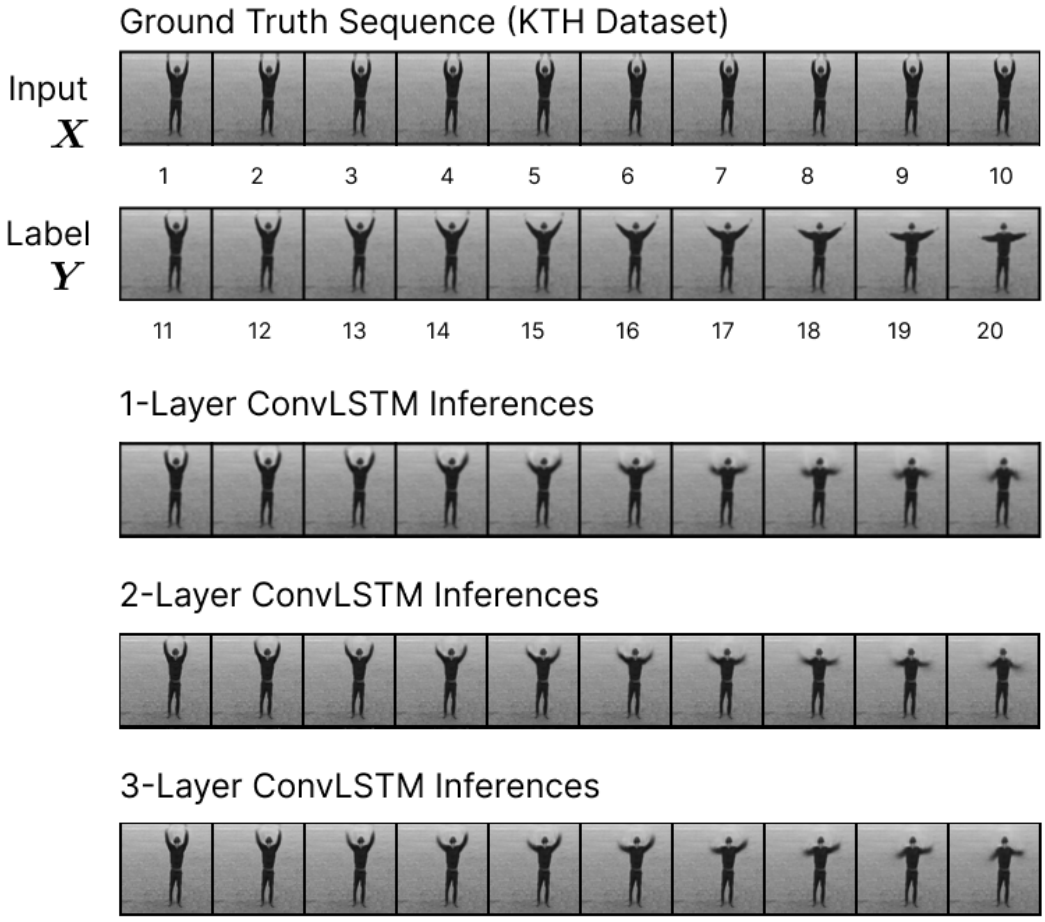

Figure 14: Convolutional LSTM Inference on KTH dataset (Sample 50)

4.2.3 BAIR

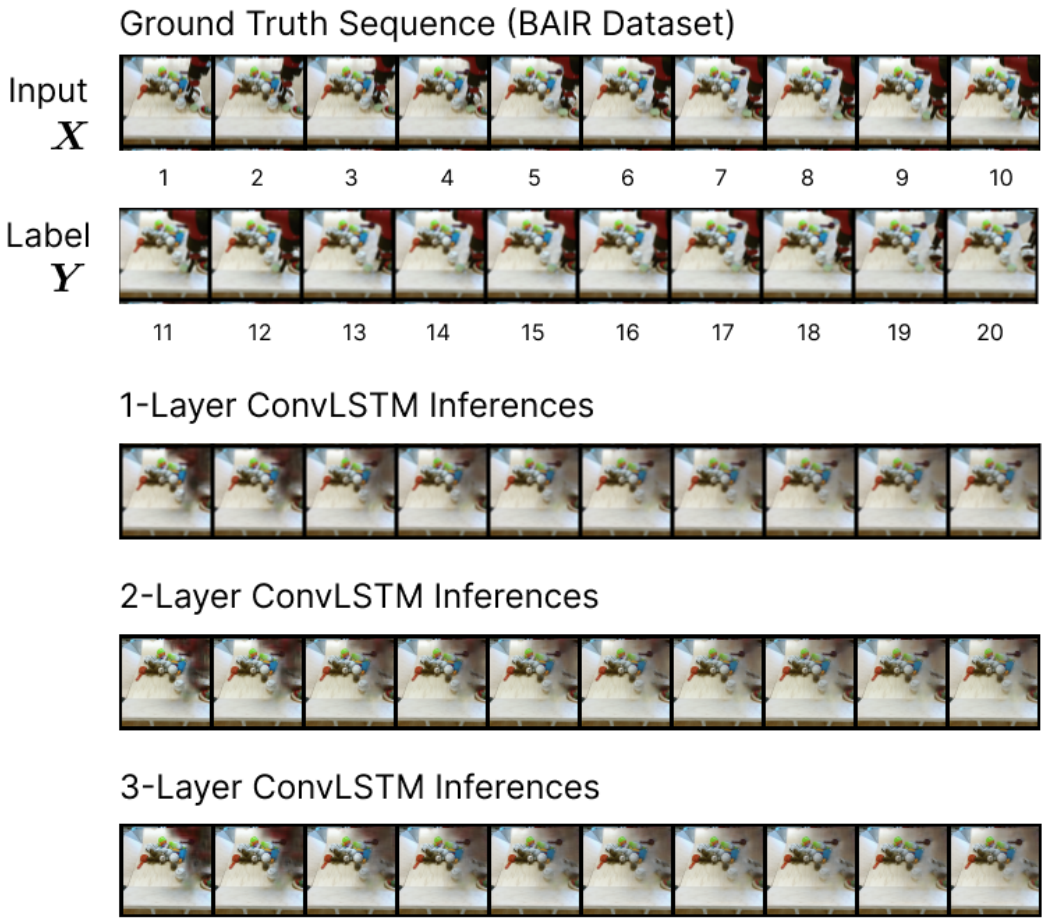

Figure 15: Convolutional LSTM Inference on BAIR dataset

5 Conclusion

References

[1] S. Aigner and M. Körner. Futuregan: Anticipating the future frames of video sequences using spatio-temporal 3d convolutions in progressively growing gans, 2018. URL https://arxiv.org/abs/1810.01325.

[2] A. Cleeremans and J. Mcclelland. Learning the structure of event sequences. *Journal of experimental psychology. General*, 120:235–53, 10 1991. doi: 10.1037//0096-3445.120.3.235.

[3] C. Finn and S. Levine. Deep visual foresight for planning robot motion, 2016. URL https://arxiv.org/abs/1610.00696.

[4] A. Hu, F. Cotter, N. Mohan, C. Gurau, and A. Kendall. Probabilistic future prediction for video scene understanding, 2020. URL https://arxiv.org/abs/2003.06409.

[5] A. X. Lee, R. Zhang, F. Ebert, P. Abbeel, C. Finn, and S. Levine. Stochastic adversarial video prediction. *CoRR*, abs/1804.01523, 2018. URL http://arxiv.org/abs/1804.01523.

[6] Z. Liu, R. A. Yeh, X. Tang, Y. Liu, and A. Agarwala. Video frame synthesis using deep voxel flow, 2017. URL https://arxiv.org/abs/1702.02463.

[7] S. Oprea, P. Martinez-Gonzalez, A. Garcia-Garcia, J. A. Castro-Vargas, S. Orts-Escolano, J. Garcia-Rodriguez, and A. Argyros. A review on deep learning techniques for video prediction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, page 1–1, 2020. ISSN 1939-3539. doi: 10.1109/tpami.2020.3045007. URL http://dx.doi.org/10.1109/TPAMI.2020.3045007.

[8] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks, 2012. URL https://arxiv.org/abs/1211.5063.

[9] A. Sherstinsky. Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. *CoRR*, abs/1808.03314, 2018. URL http://arxiv.org/abs/1808.03314.

[10] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks, 2014. URL https://arxiv.org/abs/1409.3215.

[11] Y. Yu, X. Si, C. Hu, and J. Zhang. A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures. *Neural Computation*, 31(7):1235–1270, 07 2019. ISSN 0899-7667. doi: 10.1162/neco_a_01199. URL https://doi.org/10.1162/neco_a_01199.