# Bachelorthesis

## Comparing different state-of-the-art solutions for image prediction using time-series analysis

Sören Dittrich
soeren.dittrich@uni-hildesheim.de

September 2020

**Abstract**

The thesis will compare different state-of-the-art solutions for image-/video-prediction 2.1. The main module of the solutions, which is the core aspect of this work, is the LSTM (Long short-term memory) 2.4. This module was invented by Hochreiter and Schmidhuber [7] in 1997 and is used heavily in the field of image-& video-prediction since then, e.g. in Srivastava et. al. [18]. During the time the module got many different add-ons and changes, which are described in different papers ([13], [10], [21], [20] and many more.). To have a valid comparison, I implement three different state-of-the-art solutions for image-/video-prediction ([17], [13] and [10]). All of them use the Shi et. al. ConvLSTM [17] (Or a slightly different version) as recurrent sub-module, which is changed during the experiments with another, more advanced solution named PredRNN [21]. The algorithms are re-implemented in PyTorch [12], as well as the „standard“ConvLSTM and PredRNN.

# Contents

# 1 Scientific questions

1. What different types of image prediction architectures exist?
2. How important is the choice of the recurrent module for the runtime and performance of the algorithm?

# 2 Introduction

The task of this chapter is to give the reader the basic knowledge, which is necessary to follow the rest of the thesis. In general, the reader should have basic knowledge about machine learning and neural networks. As this thesis is in the field of machine learning, more explicit neural networks and image-/video-prediction, this chapter will start giving basic knowledge about image-/video-prediction and specific neural network architectures, which are used in the thesis.

The necessary neural network architectures are Autoencoder 2.2, RNN 2.3, LSTM 2.4 and ConvLSTM 2.5, they are described briefly for the reader. Afterwards I describe the backpropagation algorithm 2.6 and BPTT 2.7 at a very basic level. Those two algorithms are the most used algorithms in training neural networks and are used in this thesis. Lastly I describe PyTorch 2.8, in which I re-implemented the baselines for this thesis.

## 2.1 Image Prediction / Video Prediction

Image-/Video-prediction is a field inside machine learning, where the key is to predict future images, given a sequence of images. The image sequence $X$ is of length $n$, $(x_0, \ldots, x_{n-1})$.

One possible use-case is the **one-frame prediction**, where one predicts $x_n$, given the the sequence $X$. Another common use-case is **multi-frame prediction**, where the key is to predict $t > 1$ many frames into the future $(x_n, \ldots, x_{n+t-1})$. This is often performed using sequence-to-sequence learning [19]. The first frames will look much better then the last frames, as ground-truth is missing, The predicted frames are only approximated, which means they contain a certain level of error, so using them as input to perform **multi-frame prediction** will increase the level of error for the following frames.

## 2.2 Autoencoder

The autoencoder is a network architecture, which consists of two neural networks chained together. The first network is called Encoder. This Encoder gets the input $x$ and outputs the code h. Often the output layer of the Encoder is named bottleneck-layer. The second network is called Decoder. It gets the code $h$ as input and outputs $x\prime$. This architecture is used for reconstruction, where $x \approx x\prime$. To prevent the architecture to simply copy the input directly to the output (which would be an interpolation and not the goal of any machine learning algorithm.), there are different techniques to have the autoencoder to instead approximate the output.

$$E(x) = h \tag{1}$$

$$D(h) = x\prime \tag{2}$$

The simplest autoencoder architecture is the so named undercomplete autoencoder [5], in which the output of the bottleneck-layer $h$ is smaller then the input $x$. Therefore the architecture needs to learn how to extract useful features from the input $x$, because it is not able to simply copy the input $x$ to the output $x\prime$, because $h$ is a reduced representation of $x$. There are many different ideas of using the autoencoder architecture, which are described more in-depth in e.g. Goodfellow et. al. [5].

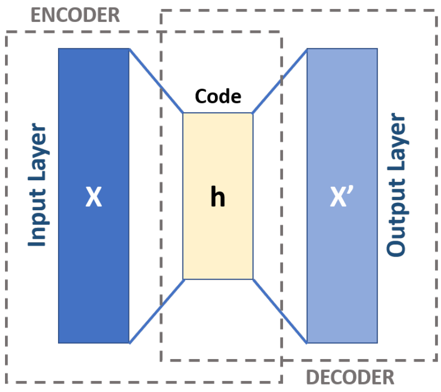



Figure 1: Autoencoder schema [11]

## 2.3 RNN

RNN (Recurrent neural network) is a network type, which is able to handle sequential data $X = (x_0, \dots, x_{t-1})$, $|X| = t$. Therefore this type of network is used for e.g. time-series analysis and image-/video-prediction 2.1.

Despite a standard feed-forward neural network, a RNN will not only get a new input at a time-step, but also the output of the RNN of the last time-step. This requires a RNN to be initialized at start, because there is no output of the last time-step available. This last time-step input is often initialized as 0. A standard feed-forward neural network looks like:

$$\hat{y} = f_\theta(x) \tag{3}$$

Every approximated output $\hat{y}$ is only dependent of the input $x$ and the computation inside the network. The RNN looks like:

$$\hat{y}^t = f_\theta(\hat{y}^{t-1}; x^t) = f_\theta(f_\theta(\hat{y}^{t-2}; x^{t-1}); x^t) = \dots \tag{4}$$

The approximated output $\hat{y}^t$ depends on the input $x^t$, but also on all previous outputs. In the literature the RNN is often schemed using a folded and an unfolded graph, to illustrate how the network architecture works.

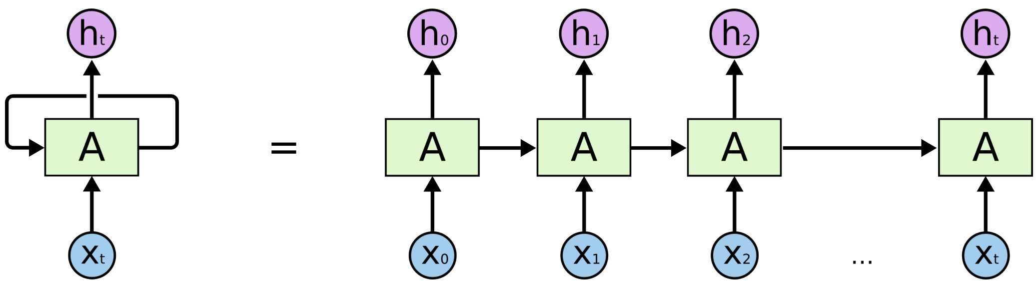



Figure 2: RNN schema. **Left:** Folded graph, **Right:** Unfolded graph [1]

The output of a RNN is often denoted as $h$ for hidden-unit, because it is not only the output of the time-step, but also the input for the next time-step.
This networks are not learned with simple backpropagation 2.6, but often with BPTT (Backpropagation through time) 2.7.

## 2.4 LSTM

LSTM (Long Short-term Memory), invented by Hochreiter and Schmidhuber [7] is a form of RNN, which avoids a critical problem of standard RNN: Saving **Long-term dependencies** [5]. The architecture consists of different submodules: An inpute-gate, forget-gate, cell-state and output-gate.

$$i_t = \sigma(w_{x_i} x_t + w_{h_i} h_{t-1} + b_i) \tag{5}$$

$$f_t = \sigma(w_{x_f} x_t + w_{h_f} h_{t-1} + b_f) \tag{6}$$

$$c_t = f_t c_{t-1} + i_t tanh(w_{x_c} x_t + w_{h_c} h_{t-1} + b_c) \tag{7}$$

$$o_t = \sigma(w_{x_o} x_t + w_{h_o} h_{t-1} + b_o) \tag{8}$$

$$h_t = o_t tanh(c_t) \tag{9}$$

$w$ is the weight of the layer
$\sigma$ the sigmoid function
$b$ the layer bias.
$h_t$ is the output, in RNN's the output is often denoted as hidden 2.3.

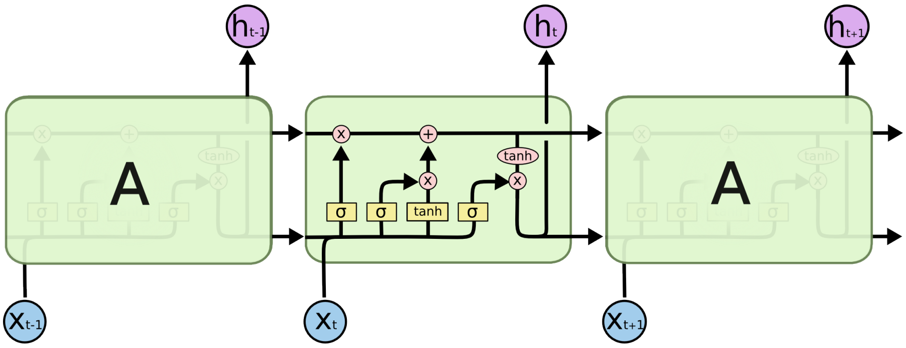



Figure 3: LSTM Architecture [1]

The given equations for the LSTM are for the very basic LSTM without peephole.
The peephole is an idea from Gers and Schmidhuber from the year 2000 [4], where they augmented the LSTM with peephole connections, which gave them an advantage in learning spike trains[2.4], because a LSTM without peephole was not able to learn those spike trains. The equations looks very similar, with only few changes:

$$i_t = \sigma(w_{x_i} x_t + w_{c_i} c_{t-1} + b_i) \tag{10}$$

$$f_t = \sigma(w_{x_f} x_t + w_{c_f} c_{t-1} + b_f) \tag{11}$$

$$c_t = f_t c_{t-1} + i_t tanh(w_{x_c} x_t + b_c) \tag{12}$$

$$o_t = \sigma(w_{x_o} x_t + w_{c_o} c_t + b_o) \tag{13}$$

$$h_t = o_t tanh(c_t) \tag{14}$$

Spike trains are time series of 0s and 1s, where 1 is defined as a spike.

## 2.5 ConvLSTM

The convolutional LSTM, invented by Shi et. al. [17] is a LSTM with peephole using convolutional layer instead of fully connected ones. Therefore the formulas look very similar to the ones in section 2.4.

$$i_t = \sigma(x_t * w_{x_i} + h_{t-1} * w_{h_i} + w_{i_b}) \tag{15}$$

$$f_t = \sigma(x_t * w_{x_f} + h_{t-1} * w_{h_f} + w_{f_b}) \tag{16}$$

$$\tilde{c_t} = tanh(x_t * w_{x_{\tilde{c}}} + h_{t-1} * w_{h_{\tilde{c}}} + w_{c_{\tilde{b}}}) \tag{17}$$

$$c_t = \tilde{c_t} \odot i_t + c_{t-1} \odot f_t \tag{18}$$

$$o_t = \sigma(x_t * w_{x_o} + h_{t-1} * w_{h_o} + w_{o_b}) \tag{19}$$

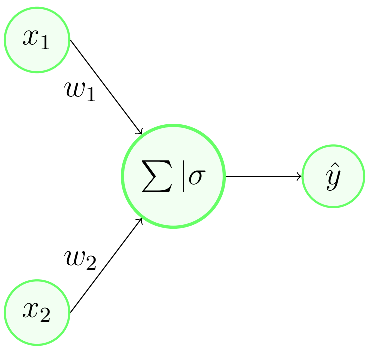
$$h_t = o_t \odot tanh(c_t) \tag{20}$$

$*$ is the commonly used sign for the convolution operation.
$\odot$ is the hadamard product (point-wise multiplication).

There is also a ConvLSTM without peephole, which is used in Patraucean et. al. [13], which is implemented in section 4.

## 2.6 Backpropagation

Backpropagation, invented in 1986 by Rumelhart et. al. [16], is the most used algorithm for training neural networks, due to it's simplicity. The algorithm should be already known to the reader, therefore I will only give a very simple example of it at the most simple neural network, the Perceptron [15].




The usage of backpropagation is very straight forward. Neural networks are representation learning algorithms, which means, that they learn the representation of the data over time, without having the need of an expert doing supervision. It only needs a valid training and testing dataset, where one has ground-truth knowledge of the output of the data. One then leverages the forward pass of the algorithm to produce our approximated output.

1. **Forward pass:**

$$\hat{y} = \sigma(\sum_{i=1}^{2} x_i w_i) \tag{21}$$

After computing the regarding output, one will compare the computed output with the ground-truth output with some kind of error-function, e.g. **MSE**:

$$L(y,\hat{y}) = \frac{1}{N}||y-\hat{y}||_2^2 = \frac{1}{N}\sum_{i=1}^{N}(y_i-\hat{y}_i)^2 \tag{22}$$

This error is then propagated back, using the chain-rule through the graph to update the weights of the network. This is done in the backward pass.

2. **Backward pass:**

$$\frac{\partial L}{\partial \hat{y}} = \frac{2}{N}\sum_{i=1}^{N}(y_i-\hat{y}_i) \tag{23}$$

$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1-\sigma(x)) \tag{24}$$

$$\frac{\partial L}{\partial \sum_{i=1}^{2} x_i w_i} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \sum_{i=1}^{2} x_i w_i} = \frac{2}{N}\sum_{i=1}^{N}(y_i-\hat{y}_i)\cdot\sigma(\sum_{i=1}^{2} x_i w_i)(1-\sigma(\sum_{i=1}^{2} x_i w_i)) \tag{25}$$

$$\frac{\partial L}{\partial w_1} = \ldots = \frac{2}{N}\sum_{i=1}^{N}(y_i-\hat{y}_i)\cdot\sigma(\sum_{i=1}^{2} x_i w_i)(1-\sigma(\sum_{i=1}^{2} x_i w_i))\cdot x_1 \tag{26}$$

3. **Update weights using Gradient Descent:**

$$w_1 = w_1^{old} - \lambda \cdot \frac{\partial L}{\partial w_1} \tag{27}$$

$\lambda$ is the learning rate and is set as a hyperparameter.

Those three steps are performed iterative, as long as the error value is higher then an artificially set value $\epsilon$.
It is also useful to use a second derivative method e.g. Newton method, instead of Gradient Descent, but in most cases Gradient Descent is used, due to it's simplicity and parallelization properties [16].

## 2.7 BPTT

BPTT (Backpropagation through time), invented by Paul Werbos [22], is a only slightly different algorithm, compared to simple backpropagation 2.6. This algorithm is explicitly designed for RNN's and as the name already denotes, able to backpropagate through the time. To make it even more clear, the algorithm does nothing more, then unfolding the graph during the backward pass and backpropagates through all unfolded time-steps performed by the recurrent module. Let's make a simple example to fully understand the idea. Therefore I will use a simple RNN architecture found in Goodfellow [5]. To have a basic comparison to the backpropagation in section 2.6, I changed the softmax function to $\sigma$, as this derivation is already known with equation 24.

1. **Forward pass:**

$$h_t = tanh(Wh_{t-1} + Ux_t + b_1) \tag{28}$$

$$o_t = Vh_t + b_2 \tag{29}$$

$$\hat{y}_t = \sigma(o_t) \tag{30}$$

$U, V, W$ are the weight matrices for input-to-hidden, hidden-to-output and hidden-to-hidden.
After computing the output, one again needs an error function, e.g. MSE (known from equation 22.)
This time, it is also necessary to have this error function for every iteration.

$$L(y, \hat{y}) = \sum_t (L_t(y_t, \hat{y}_t)) \tag{31}$$

After computing the MSE for all time-steps and the overall error, one again starts the backward pass.

2. **Backward pass:**

$$\frac{\partial L}{\partial V} = \sum_t \frac{\partial L_t}{\partial V} \tag{32}$$

$$\frac{\partial L_t}{\partial \hat{y}_t} = \frac{2}{N} \sum_{i=1}^{N} (y_t^i - \hat{y}_t^i) \tag{33}$$

$$\frac{\partial L_t}{\partial o_t} = \frac{\partial L_t}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial o_t} = \frac{2}{N} \sum_{i=1}^{N} (y_t^i - \hat{y}_t^i) \cdot \sigma(o_t)(1 - \sigma(o_t)) \tag{34}$$

$$\frac{\partial L_t}{\partial V} = \frac{\partial L_t}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial o_t} \cdot \frac{\partial o_t}{\partial V} = \frac{2}{N} \sum_{i=1}^{N} (y_t^i - \hat{y}_t^i) \cdot \sigma(o_t)(1 - \sigma(o_t)) \cdot h_t \tag{35}$$

$$\frac{\partial L}{\partial V} = \sum_t \frac{\partial L_t}{\partial V} \tag{36}$$

3. **Update weights using Gradient Descent:**

$$V = V^{old} - \lambda \cdot \frac{\partial L}{\partial V} \tag{37}$$

## 2.8 PyTorch

PyTorch [12] is a deep learning framework.

# 3 Image Prediction Architectures

This section will describe a range of state-of-the-art architectures for image prediction. Image prediction is a very broad field, but almost all state-of-the-art solutions for image prediction share one common part, the recurrent module. LSTM's are the most used modules in image prediction, as they are able to store information over a long period of time, despite the standard RNN (recurrent neural network). All algorithms described here have a different way to perform image prediction, but all use a type of LSTM to store the time-series information. It is very common for this type of papers, that the authors start their experiments by using a synthetic dataset. In the most cases this is MovingMNIST [9] and then afterwards performing tests on natural images. For natural images, the authors often use action-recognition datasets, because the camera is fixed and only objects/people in the scenery are moving. Another approach is using natural image datasets, where the camera is also moving through the scenery, for example Kitti dataset [3]. This dataset consists of natural images from different car drives through the city and residential area of Karlsruhe in Germany. In this dataset not only objects in the scenery are moving, but also the camera has a self-motion, which can be very tricky for image prediction algorithms to „understand“.

## 3.1 LSTM Autoencoder

The paper „Unsupervised Learning of Video Representations using LSTMs“by Srivastava et. al. [18] is using a LSTM 2.4 from Graves [6] in an autoencoder architecture 2.2 for image reconstruction and future image prediction. The architecture is often used as a baseline in newer and more advanced architectures, because it consists of the standard LSTM as recurrent module. Due to the fact, that the LSTM module is not able to handle multi-dimensional data as is, the images need to be reshaped at the input and also at the output again. The authors use MovingMNIST [9] as synthetic dataset, where every image is of size $(64 \times 64 \times 1)$. Therefore the image is vectorized into $(64 \cdot 64 \times 1) = (4096 \times 1)$. This MovingMNIST implementation consists of two digits inside every frame. The authors input 10 images and output the next 10 images. The model is end-to-end differentiable and trained using BPTT (backpropagation through time) [22]. For the synthetic dataset, the model is trained using cross-entropy loss with logits 2.8, for natural image datasets using MSE (mean-squared error) [23].

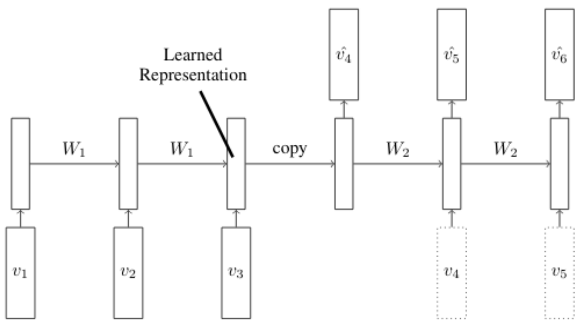



Figure 4: Architecture for future image prediction [18]

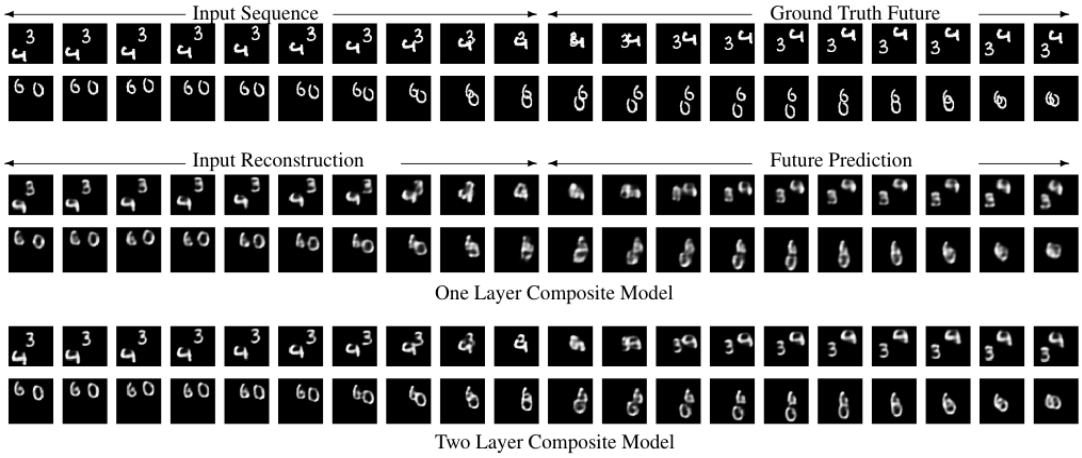



Figure 5: Results for MovingMNIST [18]

## 3.2 ConvLSTM Autoencoder

The paper „Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting“by Shi et. al. [17] is using a similar architecture as Srivastava et. al. in section 4, but instead of using the standard LSTM, they use a novel ConvLSTM 2.5. The ConvLSTM used in this architecture is with peephole **??**. This architecture outperforms the implementation of Srivastava et. al., because it „captures spatiotemporal correlations better“. This model is, same as LSTM Autoencoder 3.1 end-to-end differentiable and trained using BPTT. It also uses the cross-entropy loss with logits for the synthetic dataset (MovingMNIST) experiment. In this MovingMNIST implementation, every frame consists of three digits. As in the architecture above, the authors here input 10 images and output 10.

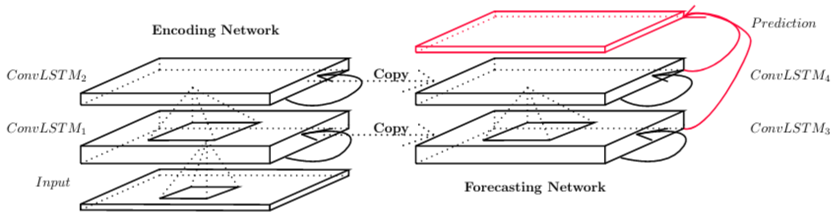



Figure 6: Future image prediction model [17]

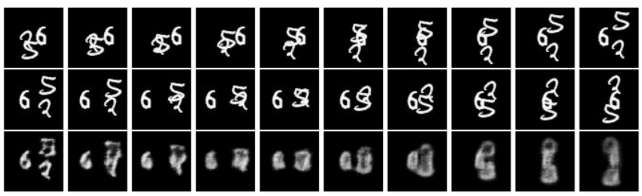


Figure 7: Results for MovingMNIST. **First Row:** 10 input images, **Second Row:** Ground truth next images, **Third Row:** Prediction of 3-layer implementation [17]

## 3.3 Spatio-temporal Video Autoencoder

The paper „Spatio-temporal Video Autoencoder With Differentiable Memory“by Patraucean et. al. [13] describes a more complex architecture, in which the authors nest a temporal autoencoder inside a spatial autoencoder. The spatial autoencoder is a simple undercomplete autoencoder 2.2, where the decoder uses nearest-neighbor upsampling to get the output image size correct. The temporal autoencoder consists of a ConvLSTM (A ConvLSTM without peephole **??**, which works as the temporal encoder and and an optical flow convolutional module, which works as the temporal decoder. The network idea is to insert the image sequence $X$, which will create an optical flow map. This optical flow map is then applied on the last given image, to shift every pixel to it's new position. This will create the next image. The idea behind this is given in „Spatial Transformer Networks“by Jadeberg et. al. [8]. The model is end-to-end differentiable and trained using BPTT. The authors also used MovingMNIST as synthetic dataset and use the cross-entropy loss with logits as reconstruction error for it. The MovingMNIST implementation is the same as in LSTM Autoencoder 3.1 (With two digits per frame.). In contrast to the other algorithms, this architecture is only capable of doing one-frame prediction as is. The authors input 19 images and output 1 image.

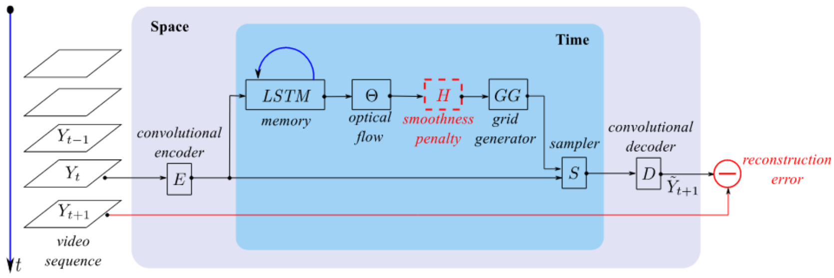



Figure 8: Spatio-temporal Video Autoencoder [13]


Figure 9: Results for MovingMNIST. **conv:** Is a simple convolutional autoencoder, without recurrent module, **fcLSTM:** LSTM Autoencoder 3.1, **cLSTM:** ConvLSTM Autoencoder 3.2, **cLSTM-flow:** Spatio-temporal Video Autoencoder with extra output of flow-map. [13]

## 3.4 PredNet

The paper „Deep Predictive Coding Networks For Video Prediction And Unsupervised Learning“by Lotter et. al. [10] composes an architecture, which is informally named **PredNet**. It describes a network architecture based on the concept of „predictive coding“[14], [2]. It is the baseline for the experiments performed in section 7. The network consists of an arbitrary amount of layers, the amount of layers (depth of the network) can be treated as a hyperparameter. Every layer consists of an input module ${A_l}^t$, prediction module ${\hat{A}_l}^t$, representation module ${R_l}^t$ and error module ${E_l}^t$. $l$ is the corresponding layer, $t$ the timestamp.

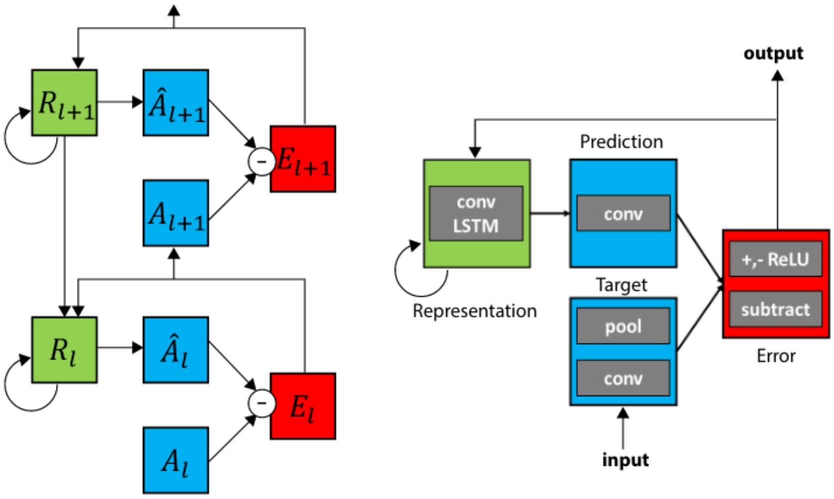



Figure 10: **Left:** PredNet architecture with two layers, **Right:** „Module operations for case of video sequences“[10]

$$ {A_l}^t = \begin{cases} x_t & l = 0 \\ MaxPool(ReLU(Conv(E_{l-1}^t))) & l > 0 \end{cases} \tag{38} $$

$$ \hat{A}_l^t = ReLU(Conv(R_l^t) \tag{39} $$

$$ E_l^t = [ReLU(\hat{A}_l^t - A_l^t); ReLU(A_l^t - \hat{A}_l^t)] \tag{40} $$

$$ R_l^t = ConvLSTM(E_l^{t-1}, R_l^{t-1}, Upsample(R_{l+1}^t) \tag{41} $$

## 3.5 PredRNN

The paper „PredRNN: Recurrent Neural Networks for Predictive Learning using Spatiotemporal LSTMs“by Wang et. al. [21].

# 4 Implementation

# 5 Methodology

# 6 Training

The training section will cover the aspects of different training types.

# 7 Experiments

The experiments performed on the implemented PredNet with ConvLSTM and PredNet with PredRNN will be described here. Also other theoretical comparisons will be covered in this section.

# 8 Discussion

# 9 Conclusion

# 10 Explanation

Erklärung über das selbstständige Verfassen von „Comparing different state-of-the-art solutions for image prediction using time-series analysis"
Ich versichere hiermit, dass ich die vorstehende Bachelorarbeit selbständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der obigen Arbeit, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen wurden, habe ich in jedem einzelnen Fall durch die Angabe der Quelle bzw. der Herkunft, auch der benutzten Sekundärliteratur, als Entlehnung kenntlich gemacht. Dies gilt auch für Zeichnungen, Skizzen, bildliche Darstellungen sowie für Quellen aus dem Internet und anderen elektronischen Text-und Datensammlungen und dergleichen. Die eingereichte Arbeit ist nicht anderweitig als Prüfungsleistung verwendet worden oder in deutscher oder in einer anderen Sprache als Veröffentlichung erschienen. Mir ist bewusst, dass wahrheitswidrige Angaben als Täuschung behandelt werden.

Datum, Ort Unterschrift

# References

[1] *Understanding LSTM Networks.* https://colah.github.io/posts/2015-08-Understanding-LSTMs/. – Accessed: 2020-07-13

[2] Friston, Karl: A Theory of Cortical Responses. In: *Philosophical transactions of the Royal Society of London. Series B, Biological sciences* 360 (2005), 05, S. 815–36

[3] Geiger, Andreas ; Lenz, Philip ; Stiller, Christoph ; Urtasun, Raquel: Vision meets Robotics: The KITTI Dataset. In: *International Journal of Robotics Research (IJRR)* (2013)

[4] Gers, F. A. ; Schmidhuber, J.: Recurrent nets that time and count. In: *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium* Bd. 3, 2000, S. 189–194 vol.3

[5] Goodfellow, Ian ; Bengio, Yoshua ; Courville, Aaron: *Deep Learning.* MIT Press, 2016. – http://www.deeplearningbook.org

[6] Graves, Alex: Generating Sequences With Recurrent Neural Networks. In: *CoRR* abs/1308.0850 (2013). – URL http://arxiv.org/abs/1308.0850

[7] Hochreiter, Sepp ; Schmidhuber, Jürgen: Long Short-term Memory. In: *Neural computation* 9 (1997), 12, S. 1735–80

[8] Jaderberg, Max ; Simonyan, Karen ; Zisserman, Andrew ; Kavukcuoglu, Koray: Spatial Transformer Networks. In: *CoRR* abs/1506.02025 (2015). – URL http://arxiv.org/abs/1506.02025

[9] Lecun, Y. ; Bottou, L. ; Bengio, Y. ; Haffner, P.: Gradient-based learning applied to document recognition. In: *Proceedings of the IEEE* 86 (1998), Nr. 11, S. 2278–2324

[10] Lotter, William ; Kreiman, Gabriel ; Cox, David D.: Deep Predictive Coding Networks for Video Prediction and Unsupervised Learning. In: *CoRR* abs/1605.08104 (2016). – URL http://arxiv.org/abs/1605.08104

[11] Massi, Michela: *Autoencoder schema — Wikipedia, The Free Encyclopedia*. 2019. – URL https://en.wikipedia.org/wiki/Autoencoder#/media/File:Autoencoder_schema.png. – [Online; accessed 21-Juli-2020]

[12] Paszke, Adam ; Gross, Sam ; Massa, Francisco ; Lerer, Adam ; Bradbury, James ; Chanan, Gregory ; Killeen, Trevor ; Lin, Zeming ; Gimelshein, Natalia ; Antiga, Luca ; Desmaison, Alban ; Kopf, Andreas ; Yang, Edward ; DeVito, Zachary ; Raison, Martin ; Tejani, Alykhan ; Chilamkurthy, Sasank ; Steiner, Benoit ; Fang, Lu ; Bai, Junjie ; Chintala, Soumith: PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: Wallach, H. (Hrsg.) ; Larochelle, H. (Hrsg.) ; Beygelzimer, A. (Hrsg.) ; Alché-Buc, F. d(Hrsg.) ; Fox, E. (Hrsg.) ; Garnett, R. (Hrsg.): *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, S. 8024–8035. – URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[13] Patraucean, Viorica ; Handa, Ankur ; Cipolla, Roberto: Spatio-temporal video autoencoder with differentiable memory. In: *CoRR* abs/1511.06309 (2015). – URL http://arxiv.org/abs/1511.06309

[14] Rao, Rajesh P. N. ; Ballard, Dana H.: Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. In: *Nature Neuroscience* 2 (1999), Jan, Nr. 1, S. 79–87. – URL https://doi.org/10.1038/4580. – ISSN 1546-1726

[15] Rosenblatt, Frank: The Perceptron — A Perceiving and Recognizing Automaton. In: *Tech. Rep. 85-460-1* (1957)

[16] Rumelhart, David E. ; Hinton, Geoffrey E. ; Williams, Ronald J.: Learning representations by back-propagating errors. In: *Nature* 323 (1986), Oct, Nr. 6088, S. 533–536. – URL https://doi.org/10.1038/323533a0. – ISSN 1476-4687

[17] SHI, Xingjian ; Chen, Zhourong ; Wang, Hao ; Yeung, Dit-Yan ; Wong, Wai-kin ; WOO, Wang-chun: Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting. In: Cortes, C. (Hrsg.) ; Lawrence, N. D. (Hrsg.) ; Lee, D. D. (Hrsg.) ; Sugiyama, M. (Hrsg.) ; Garnett, R. (Hrsg.): *Advances in Neural Information Processing Systems 28*. Curran Associates, Inc., 2015, S. 802–810. – URL http://papers.nips.cc/paper/5955-convolutional-lstm-network-a-machine-learning-approach-for-precipitation-nowca pdf

[18] Srivastava, Nitish ; Mansimov, Elman ; Salakhutdinov, Ruslan: Unsupervised Learning of Video Representations using LSTMs. In: *CoRR* abs/1502.04681 (2015). – URL http://arxiv.org/abs/1502.04681

[19] Sutskever, Ilya ; Vinyals, Oriol ; Le, Quoc V.: Sequence to Sequence Learning with Neural Networks. In: *CoRR* abs/1409.3215 (2014). – URL http://arxiv.org/abs/1409.3215

[20] Wang, Yunbo ; Gao, Zhifeng ; Long, Mingsheng ; Wang, Jianmin ; Yu, Philip S.: PredRNN++: Towards A Resolution of the Deep-in-Time Dilemma in Spatiotemporal Predictive Learning. In: *CoRR* abs/1804.06300 (2018). – URL `http://arxiv.org/abs/1804.06300`

[21] Wang, Yunbo ; Long, Mingsheng ; Wang, Jianmin ; Gao, Zhifeng ; Yu, Philip S.: PredRNN: Recurrent Neural Networks for Predictive Learning using Spatiotemporal LSTMs. In: *NIPS*, 2017

[22] Werbos, Paul: Backpropagation through time: what it does and how to do it. In: *Proceedings of the IEEE* 78 (1990), 11, S. 1550 – 1560

[23] Zhao, H. ; Gallo, O. ; Frosio, I. ; Kautz, J.: Loss Functions for Image Restoration With Neural Networks. In: *IEEE Transactions on Computational Imaging* 3 (2017), Nr. 1, S. 47–57