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

Sören Dittrich

University of Hildesheim

Summerterm 2020

Table of contents

Contents

Scientific questions

1. What different types of image-/video-prediction architectures exist?

Scientific questions

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

Scientific questions

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

Contents

Introduction

- Overview of necessary machine learning theory

Introduction

- Overview of necessary machine learning theory
- Overview of different state-of-the-art solutions for image-/video-prediction

Introduction

- Overview of necessary machine learning theory
- Overview of different state-of-the-art solutions for image-/video-prediction
- Implementation of three image-/video-prediction algorithms in PyTorch [10]

Introduction

- Overview of necessary machine learning theory
- Overview of different state-of-the-art solutions for image-/video-prediction
- Implementation of three image-/video-prediction algorithms in PyTorch [10]
- Change of recurrent module during experiments to achieve performance improvement

Introduction

- Overview of necessary machine learning theory
- Overview of different state-of-the-art solutions for image-/video-prediction
- Implementation of three image-/video-prediction algorithms in PyTorch [10]
- Change of recurrent module during experiments to achieve performance improvement
- Discussion of the overview and the performed experiments

Introduction

- Overview of necessary machine learning theory
- Overview of different state-of-the-art solutions for image-/video-prediction
- Implementation of three image-/video-prediction algorithms in PyTorch [10]
- Change of recurrent module during experiments to achieve performance improvement
- Discussion of the overview and the performed experiments
- Close with a conclusion

Contents

Image Prediction

- Field inside machine learning / computer vision

Image Prediction

- Field inside machine learning / computer vision
- Predict future image/s, given sequence of images

Image Prediction

- Field inside machine learning / computer vision
- Predict future image/s, given sequence of images
- X the image sequence of length n

Image Prediction

- Field inside machine learning / computer vision
- Predict future image/s, given sequence of images
- X the image sequence of length n
- with $X = (x_0, \ldots, x_{n-1})$

Image Prediction

- Field inside machine learning / computer vision
- Predict future image/s, given sequence of images
- X the image sequence of length n
- with $X = (x_0, \ldots, x_{n-1})$
- Two possible use-cases

Image Prediction

- Field inside machine learning / computer vision
- Predict future image/s, given sequence of images
- X the image sequence of length n
- with $X = (x_0, \ldots, x_{n-1})$
- Two possible use-cases
    - One-frame prediction

Image Prediction

- Field inside machine learning / computer vision
- Predict future image/s, given sequence of images
- X the image sequence of length n
- with $X = (x_0, \ldots, x_{n-1})$
- Two possible use-cases
  - One-frame prediction
    - Predicting x_n

Image Prediction

- Field inside machine learning / computer vision
- Predict future image/s, given sequence of images
- X the image sequence of length n
- with $X = (x_0, \ldots, x_{n-1})$
- Two possible use-cases
  - One-frame prediction
    - Predicting x_n
  - Multi-frame prediction

Image Prediction

- Field inside machine learning / computer vision
- Predict future image/s, given sequence of images
- X the image sequence of length n
- with $X = (x_0, \ldots, x_{n-1})$
- Two possible use-cases
    - One-frame prediction
        - Predicting x_n
    - Multi-frame prediction
        - Predict $t > 1$ frames into the future $(x_n, \ldots, x_{n+t-1})$

Image Prediction

- Field inside machine learning / computer vision
- Predict future image/s, given sequence of images
- X the image sequence of length n
- with $X = (x_0, \ldots, x_{n-1})$
- Two possible use-cases
  - One-frame prediction
    - Predicting x_n
  - Multi-frame prediction
    - Predict $t > 1$ frames into the future $(x_n, \ldots, x_{n+t-1})$
    - Often it is one-frame prediction in a feedback loop

Image Prediction

- Field inside machine learning / computer vision
- Predict future image/s, given sequence of images
- X the image sequence of length n
- with $X = (x_0, \ldots, x_{n-1})$
- Two possible use-cases
  - One-frame prediction
    - Predicting x_n
  - Multi-frame prediction
    - Predict $t > 1$ frames into the future $(x_n, \ldots, x_{n+t-1})$
    - Often it is one-frame prediction in a feedback loop
    - Propagate the error $\rightarrow$ Greater error in later images

Autoencoder

- Two networks chained together

[1] h is the so named **code**. Output layer is named **bottleneck layer**.

- Two networks chained together
  - Encoder

[1] *h* is the so named **code**. Output layer is named **bottleneck layer**.

Autoencoder

- Two networks chained together
  - Encoder
    - Input is x

[1] h is the so named **code**. Output layer is named **bottleneck layer**.

Autoencoder

- Two networks chained together
  - Encoder
    - Input is x
    - Output is h[1]

[1] h is the so named **code**. Output layer is named **bottleneck layer**.

Autoencoder

- Two networks chained together
  - Encoder
    - Input is x
    - Output is h[1]
    - $E(x) = h$

[1] h is the so named **code**. Output layer is named **bottleneck layer**.

Autoencoder

- Two networks chained together
  - Encoder
    - Input is x
    - Output is h[1]
    - $E(x) = h$
  - Decoder

[1] h is the so named **code**. Output layer is named **bottleneck layer**.

Autoencoder

- Two networks chained together
  - Encoder
    - Input is x
    - Output is h[1]
    - $E(x) = h$
  - Decoder
    - Input is h

[1] h is the so named **code**. Output layer is named **bottleneck layer**.

Autoencoder

- Two networks chained together
  - Encoder
    - Input is x
    - Output is h[1]
    - $E(x) = h$
  - Decoder
    - Input is h
    - Output is $x\prime$

[1] h is the so named **code**. Output layer is named **bottleneck layer**.

Autoencoder

- Two networks chained together
  - Encoder
    - Input is x
    - Output is h[1]
    - $E(x) = h$
  - Decoder
    - Input is h
    - Output is $x\prime$
    - $D(h) = x\prime$

[1] h is the so named **code**. Output layer is named **bottleneck layer**.

Autoencoder

- Two networks chained together
  - Encoder
    - Input is x
    - Output is h[1]
    - $E(x) = h$
  - Decoder
    - Input is h
    - Output is $x\prime$
    - $D(h) = x\prime$
- Used for reconstruction $x \approx x\prime$

[1] h is the so named **code**. Output layer is named **bottleneck layer**.

Autoencoder

- Two networks chained together
  - Encoder
    - Input is x
    - Output is h[1]
    - $E(x) = h$
  - Decoder
    - Input is h
    - Output is $x\prime$
    - $D(h) = x\prime$
- Used for reconstruction $x \approx x\prime$
- Important is to prevent the network to simply copy x to $x\prime$ (Interpolation)

[1] h is the so named **code**. Output layer is named **bottleneck layer**.

Autoencoder

- Two networks chained together
  - Encoder
    - Input is x
    - Output is h[1]
    - $E(x) = h$
  - Decoder
    - Input is h
    - Output is $x\prime$
    - $D(h) = x\prime$
- Used for reconstruction $x \approx x\prime$
- Important is to prevent the network to simply copy x to $x\prime$ (Interpolation)
- Simplest architecture is the undercomplete autoencoder

[1] h is the so named **code**. Output layer is named **bottleneck layer**.

Autoencoder

- Two networks chained together
  - Encoder
    - Input is x
    - Output is h[1]
    - $E(x) = h$
  - Decoder
    - Input is h
    - Output is $x\prime$
    - $D(h) = x\prime$
- Used for reconstruction $x \approx x\prime$
- Important is to prevent the network to simply copy x to $x\prime$ (Interpolation)
- Simplest architecture is the undercomplete autoencoder
  - Code smaller then input

[1] h is the so named **code**. Output layer is named **bottleneck layer**.

Autoencoder

- Two networks chained together
  - Encoder
    - Input is x
    - Output is h[1]
    - $E(x) = h$
  - Decoder
    - Input is h
    - Output is $x\prime$
    - $D(h) = x\prime$
- Used for reconstruction $x \approx x\prime$
- Important is to prevent the network to simply copy x to $x\prime$ (Interpolation)
- Simplest architecture is the undercomplete autoencoder
  - Code smaller then input
  - Network needs to distinguish between useful and obsolete

[1] h is the so named **code**. Output layer is named **bottleneck layer**.

Autoencoder

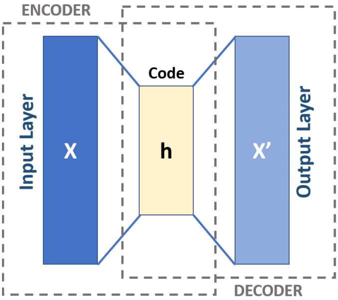


Figure: Autoencoder schema [9]

CNN

- Convolutional Neural Network

CNN

- Convolutional Neural Network
- Consists of three stages

CNN

- Convolutional Neural Network
- Consists of three stages
  1. Convolutional layer

CNN

- Convolutional Neural Network
- Consists of three stages
  1. Convolutional layer
  2. Non-linearity (ReLU, sigmoid, . . .)

CNN

- Convolutional Neural Network
- Consists of three stages
  1. Convolutional layer
  2. Non-linearity (ReLU, sigmoid, ...)
  3. Pooling layer

CNN

- Convolutional Neural Network
- Consists of three stages
  1. Convolutional layer
  2. Non-linearity (ReLU, sigmoid, . . .)
  3. Pooling layer

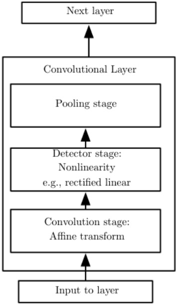


Figure: Stages of a CNN [4]

CNN (First stage)

- Discrete convolutional operation

CNN (First stage)

- Discrete convolutional operation
- $(I * K)(i, j) = \sum_m \sum_n I(m, n) K(i - m, j - n)$

CNN (First stage)

- Discrete convolutional operation
- $(I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)$

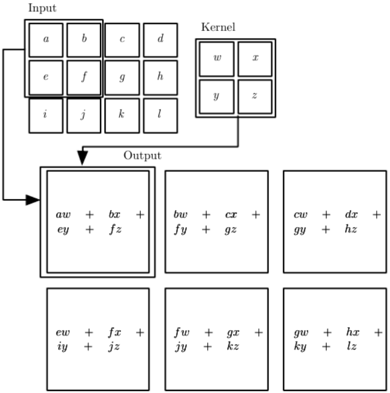


Figure: Two dimensional convolutional operation [4]

CNN (Second stage)

- Non-linearity layer

CNN (Second stage)

- Non-linearity layer
- ReLU: $R(x) = max(0, x)$

CNN (Second stage)

- Non-linearity layer
- ReLU: $R(x) = max(0, x)$
- Sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}}$

CNN (Third stage)

- Pooling layer

CNN (Third stage)

- Pooling layer
- Aggregate output / Making output invariant to input

CNN (Third stage)

- Pooling layer
- Aggregate output / Making output invariant to input
- E.g. Max-Pooling

CNN (Third stage)

- Pooling layer
- Aggregate output / Making output invariant to input
- E.g. Max-Pooling

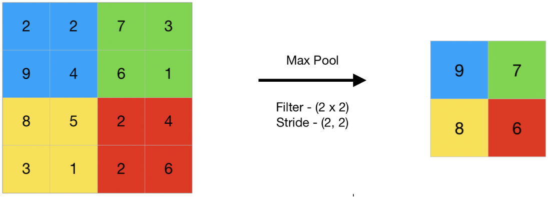


Figure: Max-Pooling (Filter := Kernel)[1]

RNN

- Recurrent Neural Network

RNN

- Recurrent Neural Network
- Handling sequential data

RNN

- Recurrent Neural Network
- Handling sequential data
- $X = (x_0, \ldots, x_{t-1})$, $|X| = t$

RNN

- Recurrent Neural Network
- Handling sequential data
- $X = (x_0, \ldots, x_{t-1})$, $|X| = t$
- Used for e.g. time-series analysis, image-/video-prediction

RNN

- Recurrent Neural Network
- Handling sequential data
- $X = (x_0, \ldots, x_{t-1})$, $|X| = t$
- Used for e.g. time-series analysis, image-/video-prediction
- Gets output of last iteration as „additional“input

RNN

- Recurrent Neural Network
- Handling sequential data
- $X = (x_0, \dots, x_{t-1})$, $|X| = t$
- Used for e.g. time-series analysis, image-/video-prediction
- Gets output of last iteration as „additional“input
- $\hat{y}^t = f_\theta(\hat{y}^{t-1}; x^t) = f_\theta(f_\theta(\hat{y}^{t-2}; x^{t-1}); x^t) = \dots$

RNN

- Recurrent Neural Network
- Handling sequential data
- $X = (x_0, \ldots, x_{t-1})$, $|X| = t$
- Used for e.g. time-series analysis, image-/video-prediction
- Gets output of last iteration as „additional"input
- $\hat{y}^t = f_\theta(\hat{y}^{t-1}; x^t) = f_\theta(f_\theta(\hat{y}^{t-2}; x^{t-1}); x^t) = \ldots$
- Requires $\hat{y}^0$ to be initialized (Often done with zero)

RNN

- Recurrent Neural Network
- Handling sequential data
- $X = (x_0, \dots, x_{t-1})$, $|X| = t$
- Used for e.g. time-series analysis, image-/video-prediction
- Gets output of last iteration as „additional“input
- $\hat{y}^t = f_\theta(\hat{y}^{t-1}; x^t) = f_\theta(f_\theta(\hat{y}^{t-2}; x^{t-1}); x^t) = \dots$
- Requires $\hat{y}^0$ to be initialized (Often done with zero)
- Typically trained using BPTT

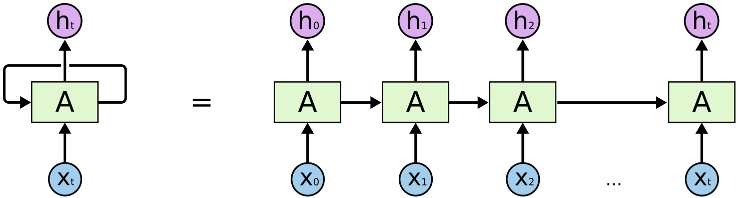


Figure: RNN schema. **Left:** Folded graph, **Right:** Unfolded graph [2]

LSTM

- Long Short-term Memory

LSTM

- Long Short-term Memory
- Invented by Hochreiter and Schmidhuber [5]

LSTM

- Long Short-term Memory
- Invented by Hochreiter and Schmidhuber [5]
- Avoids critical problem of standard RNN: Saving long-term dependencies

LSTM

- Long Short-term Memory
- Invented by Hochreiter and Schmidhuber [5]
- Avoids critical problem of standard RNN: Saving long-term dependencies

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

LSTM

- Long Short-term Memory
- Invented by Hochreiter and Schmidhuber [5]
- Avoids critical problem of standard RNN: Saving long-term dependencies

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

$$f_t = \sigma(w_{x_f} x_t + w_{h_f} h_{t-1} + b_f) \tag{2}$$

LSTM

- Long Short-term Memory
- Invented by Hochreiter and Schmidhuber [5]
- Avoids critical problem of standard RNN: Saving long-term dependencies

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

$$f_t = \sigma(w_{x_f} x_t + w_{h_f} h_{t-1} + b_f) \tag{2}$$

$$c_t = f_t c_{t-1} + i_t tanh(w_{x_c} x_t + w_{h_c} h_{t-1} + b_c) \tag{3}$$

LSTM

- Long Short-term Memory
- Invented by Hochreiter and Schmidhuber [5]
- Avoids critical problem of standard RNN: Saving long-term dependencies

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

$$f_t = \sigma(w_{x_f} x_t + w_{h_f} h_{t-1} + b_f) \tag{2}$$

$$c_t = f_t c_{t-1} + i_t tanh(w_{x_c} x_t + w_{h_c} h_{t-1} + b_c) \tag{3}$$

$$o_t = \sigma(w_{x_o} x_t + w_{h_o} h_{t-1} + b_o) \tag{4}$$

LSTM

- Long Short-term Memory
- Invented by Hochreiter and Schmidhuber [5]
- Avoids critical problem of standard RNN: Saving long-term dependencies

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

$$f_t = \sigma(w_{x_f} x_t + w_{h_f} h_{t-1} + b_f) \tag{2}$$

$$c_t = f_t c_{t-1} + i_t tanh(w_{x_c} x_t + w_{h_c} h_{t-1} + b_c) \tag{3}$$

$$o_t = \sigma(w_{x_o} x_t + w_{h_o} h_{t-1} + b_o) \tag{4}$$

$$h_t = o_t tanh(c_t) \tag{5}$$

LSTM

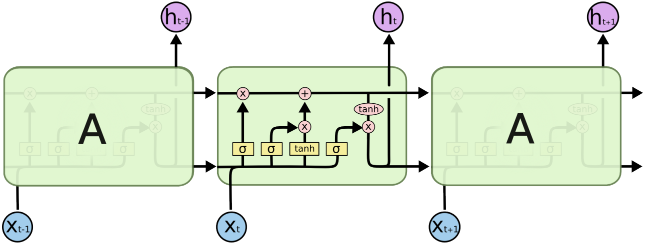


Figure: LSTM Architecture [2]

LSTM

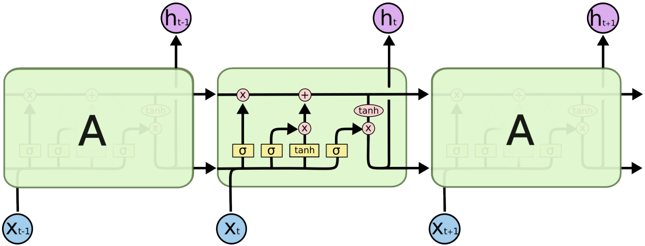


Figure: LSTM Architecture [2]

- Top horizontal line: Cell-state

ConvLSTM

- Invented by Shi et al. [14]

ConvLSTM

- Invented by Shi et al. [14]
- LSTM with peephole using convlayer

ConvLSTM

- Invented by Shi et al. [14]
- LSTM with peephole using convlayer

$$i_t = \sigma(x_t * w_{x_i} + h_{t-1} * w_{h_i} + w_{i_b}) \tag{6}$$

ConvLSTM

- Invented by Shi et al. [14]
- LSTM with peephole using convlayer

$$i_t = \sigma(x_t * w_{x_i} + h_{t-1} * w_{h_i} + w_{i_b}) \tag{6}$$

$$f_t = \sigma(x_t * w_{x_f} + h_{t-1} * w_{h_f} + w_{f_b}) \tag{7}$$

ConvLSTM

- Invented by Shi et al. [14]
- LSTM with peephole using convlayer

$$i_t = \sigma(x_t * w_{x_i} + h_{t-1} * w_{h_i} + w_{i_b}) \tag{6}$$

$$f_t = \sigma(x_t * w_{x_f} + h_{t-1} * w_{h_f} + w_{f_b}) \tag{7}$$

$$\tilde{c}_t = tanh(x_t * w_{x_{\tilde{c}}} + h_{t-1} * w_{h_{\tilde{c}}} + w_{\tilde{c}_b}) \tag{8}$$

ConvLSTM

- Invented by Shi et al. [14]
- LSTM with peephole using convlayer

$$i_t = \sigma(x_t * w_{x_i} + h_{t-1} * w_{h_i} + w_{i_b}) \tag{6}$$

$$f_t = \sigma(x_t * w_{x_f} + h_{t-1} * w_{h_f} + w_{f_b}) \tag{7}$$

$$\tilde{c}_t = tanh(x_t * w_{x_{\tilde{c}}} + h_{t-1} * w_{h_{\tilde{c}}} + w_{\tilde{c}_b}) \tag{8}$$

$$c_t = \tilde{c}_t \odot i_t + c_{t-1} \odot f_t \tag{9}$$

ConvLSTM

- Invented by Shi et al. [14]
- LSTM with peephole using convlayer

$$i_t = \sigma(x_t * w_{x_i} + h_{t-1} * w_{h_i} + w_{i_b}) \tag{6}$$

$$f_t = \sigma(x_t * w_{x_f} + h_{t-1} * w_{h_f} + w_{f_b}) \tag{7}$$

$$\tilde{c}_t = tanh(x_t * w_{x_{\tilde{c}}} + h_{t-1} * w_{h_{\tilde{c}}} + w_{\tilde{c}_b}) \tag{8}$$

$$c_t = \tilde{c}_t \odot i_t + c_{t-1} \odot f_t \tag{9}$$

$$o_t = \sigma(x_t * w_{x_o} + h_{t-1} * w_{h_o} + w_{o_b}) \tag{10}$$

ConvLSTM

- Invented by Shi et al. [14]
- LSTM with peephole using convlayer

$$i_t = \sigma(x_t * w_{x_i} + h_{t-1} * w_{h_i} + w_{i_b}) \tag{6}$$

$$f_t = \sigma(x_t * w_{x_f} + h_{t-1} * w_{h_f} + w_{f_b}) \tag{7}$$

$$\tilde{c}_t = tanh(x_t * w_{x_{\tilde{c}}} + h_{t-1} * w_{h_{\tilde{c}}} + w_{\tilde{c}_b}) \tag{8}$$

$$c_t = \tilde{c}_t \odot i_t + c_{t-1} \odot f_t \tag{9}$$

$$o_t = \sigma(x_t * w_{x_o} + h_{t-1} * w_{h_o} + w_{o_b}) \tag{10}$$

$$h_t = o_t \odot tanh(c_t) \tag{11}$$

Backpropagation

- Backpropagation was invented by Rumelhart et al. [13] in 1986.

Backpropagation

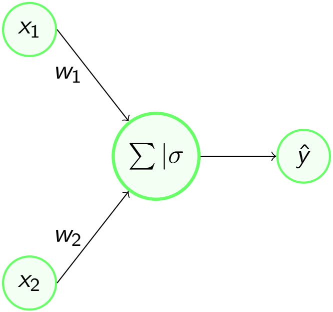
- Backpropagation was invented by Rumelhart et al. [13] in 1986.
- Most used algorithm for training neural nets.

Backpropagation

- Backpropagation was invented by Rumelhart et al. [13] in 1986.
- Most used algorithm for training neural nets.
- Simple example on Perceptron [12].

Backpropagation

- Backpropagation was invented by Rumelhart et al. [13] in 1986.
- Most used algorithm for training neural nets.
- Simple example on Perceptron [12].



Backpropagation Forward Pass

Forward Pass:

Backpropagation Forward Pass

Forward Pass:

- $$\hat{y} = \sigma(\sum_{i=1}^{2} x_i w_i) \tag{12}$$

Backpropagation Forward Pass

Forward Pass:

- $$\hat{y} = \sigma(\sum_{i=1}^{2} x_i w_i) \tag{12}$$
- σ is the sigmoid function

Backpropagation Forward Pass

Forward Pass:

- $$\hat{y} = \sigma(\sum_{i=1}^{2} x_i w_i) \tag{12}$$
- σ is the sigmoid function
- $$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{13}$$

Backpropagation Forward Pass

Forward Pass:

- $$\hat{y} = \sigma(\sum_{i=1}^{2} x_i w_i) \tag{12}$$
- σ is the sigmoid function
- $$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{13}$$
- Computing the error with e.g. **MSE**

Backpropagation Forward Pass

Forward Pass:

- $$\hat{y} = \sigma\left(\sum_{i=1}^{2} x_i w_i\right) \quad (12)$$
- σ is the sigmoid function
- $$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (13)$$
- Computing the error with e.g. **MSE**
- $$L(y, \hat{y}) = \frac{1}{N} ||y - \hat{y}||_2^2 = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \quad (14)$$

Backprojection Backward Pass

Backward Pass:

Backpropagation Backward Pass

Backward Pass:

- $$\frac{\partial L}{\partial \hat{y}} = \frac{2}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i) \tag{15}$$

Backpropagation Backward Pass

Backward Pass:

- $$\frac{\partial L}{\partial \hat{y}} = \frac{2}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i) \tag{15}$$

- $$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x)) \tag{16}$$

Backpropagation Backward Pass

Backward Pass:

- $$\frac{\partial L}{\partial \hat{y}} = \frac{2}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i) \tag{15}$$

- $$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1-\sigma(x)) \tag{16}$$

- $$\frac{\partial L}{\partial \sum_{i=1}^{2} x_i w_i} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \sum_{i=1}^{2} x_i w_i} = \frac{2}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)\cdot\sigma(\sum_{i=1}^{2} x_i w_i)(1-\sigma(\sum_{i=1}^{2} x_i w_i)) \tag{17}$$

Backpropagation Backward Pass

Backward Pass:

- $$\frac{\partial L}{\partial \hat{y}} = \frac{2}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i) \tag{15}$$
- $$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x)) \tag{16}$$
- $$\frac{\partial L}{\partial \sum_{i=1}^{2} x_i w_i} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \sum_{i=1}^{2} x_i w_i} = \frac{2}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i) \cdot \sigma(\sum_{i=1}^{2} x_i w_i)(1 - \sigma(\sum_{i=1}^{2} x_i w_i)) \tag{17}$$
- $$\frac{\partial L}{\partial w_1} = \ldots = \frac{2}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i) \cdot \sigma(\sum_{i=1}^{2} x_i w_i)(1 - \sigma(\sum_{i=1}^{2} x_i w_i)) \cdot x_1 \tag{18}$$

Backpropagation Update

Update weights using Gradient Descent

Backpropagation Update

Update weights using Gradient Descent

- $$w_1 = w_1^{old} - \lambda \cdot \frac{\partial L}{\partial w_1} \tag{19}$$

Backpropagation Update

Update weights using Gradient Descent

- $$w_1 = w_1^{old} - \lambda \cdot \frac{\partial L}{\partial w_1} \tag{19}$$
- λ is the learning rate

Backpropagation Update

Update weights using Gradient Descent

- $$w_1 = w_1^{old} - \lambda \cdot \frac{\partial L}{\partial w_1} \tag{19}$$
- λ is the learning rate
- All steps are performed iterative, until convergence

Backpropagation Update

Update weights using Gradient Descent

- $$w_1 = w_1^{old} - \lambda \cdot \frac{\partial L}{\partial w_1} \tag{19}$$
- λ is the learning rate
- All steps are performed iterative, until convergence
- It would be also possible to use e.g. Newton instead of Gradient Descent

Backpropagation Update

Update weights using Gradient Descent

- $$w_1 = w_1^{old} - \lambda \cdot \frac{\partial L}{\partial w_1} \tag{19}$$
- λ is the learning rate
- All steps are performed iterative, until convergence
- It would be also possible to use e.g. Newton instead of Gradient Descent
    - Gradient Descent is used more, because of its simplicity

Backpropagation Update

Update weights using Gradient Descent

- $$w_1 = w_1^{old} - \lambda \cdot \frac{\partial L}{\partial w_1} \tag{19}$$
- λ is the learning rate
- All steps are performed iterative, until convergence
- It would be also possible to use e.g. Newton instead of Gradient Descent
  - Gradient Descent is used more, because of its simplicity
  - and because of its parallelism properties [13]

BPTT

- Backpropagation through time

BPTT

- Backpropagation through time
- Invented by Paul Werbos [17]

BPTT

- Backpropagation through time
- Invented by Paul Werbos [17]
- Unfolding graph during backpass and propagate through the steps

BPTT

- Backpropagation through time
- Invented by Paul Werbos [17]
- Unfolding graph during backpass and propagate through the steps
- Simple example on RNN found in Goodfellow [4]

BPTT Forward Pass

Forward Pass:

BPTT Forward Pass

Forward Pass:

-
$$h_t = tanh(Wh_{t-1} + Ux_t + b_1) \tag{20}$$
$$o_t = Vh_t + b_2 \tag{21}$$
$$\hat{y}_t = \sigma(o_t) \tag{22}$$

BPTT Forward Pass

Forward Pass:

- $$h_t = tanh(Wh_{t-1} + Ux_t + b_1) \quad (20)$$
  $$o_t = Vh_t + b_2 \quad (21)$$
  $$\hat{y}_t = \sigma(o_t) \quad (22)$$
- U, V, W weight matrices for input-to-hidden, hidden-to-output, hidden-to-hidden

BPTT Forward Pass

Forward Pass:

- $$h_t = tanh(Wh_{t-1} + Ux_t + b_1) \tag{20}$$
  $$o_t = Vh_t + b_2 \tag{21}$$
  $$\hat{y}_t = \sigma(o_t) \tag{22}$$
- U, V, W weight matrices for input-to-hidden, hidden-to-output, hidden-to-hidden
- Computing the error with e.g. **MSE** for every iteration

BPTT Forward Pass

Forward Pass:

- $$h_t = tanh(Wh_{t-1} + Ux_t + b_1) \quad (20)$$
  $$o_t = Vh_t + b_2 \quad (21)$$
  $$\hat{y}_t = \sigma(o_t) \quad (22)$$
- U, V, W weight matrices for input-to-hidden, hidden-to-output, hidden-to-hidden
- Computing the error with e.g. **MSE** for every iteration
- $$L(y, \hat{y}) = \sum_t (L_t(y_t, \hat{y}_t)) \quad (23)$$

BPTT Backward Pass

Backward Pass:

BPTT Backward Pass

Backward Pass:

- $$\frac{\partial L}{\partial V} = \sum_t \frac{\partial L_t}{\partial V} \tag{24}$$

BPTT Backward Pass

Backward Pass:

- $$\frac{\partial L}{\partial V} = \sum_t \frac{\partial L_t}{\partial V} \tag{24}$$

- $$\frac{\partial L_t}{\partial \hat{y}_t} = \frac{2}{N} \sum_{i=1}^{N} (y_t^i - \hat{y}_t^i) \tag{25}$$

BPTT Backward Pass

Backward Pass:

- $$\frac{\partial L}{\partial V} = \sum_t \frac{\partial L_t}{\partial V} \tag{24}$$
- $$\frac{\partial L_t}{\partial \hat{y}_t} = \frac{2}{N} \sum_{i=1}^{N} (y_t^i - \hat{y}_t^i) \tag{25}$$
- $$\frac{\partial L_t}{\partial o_t} = \frac{\partial L_t}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial o_t} = \frac{2}{N} \sum_{i=1}^{N} (y_t^i - \hat{y}_t^i) \cdot \sigma(o_t)(1 - \sigma(o_t)) \tag{26}$$

BPTT Backward Pass

Backward Pass:

- $$\frac{\partial L}{\partial V} = \sum_t \frac{\partial L_t}{\partial V} \tag{24}$$

- $$\frac{\partial L_t}{\partial \hat{y}_t} = \frac{2}{N} \sum_{i=1}^{N} (y_t^i - \hat{y}_t^i) \tag{25}$$

- $$\frac{\partial L_t}{\partial o_t} = \frac{\partial L_t}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial o_t} = \frac{2}{N} \sum_{i=1}^{N} (y_t^i - \hat{y}_t^i) \cdot \sigma(o_t)(1 - \sigma(o_t)) \tag{26}$$

- $$\frac{\partial L_t}{\partial V} = \frac{\partial L_t}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial o_t} \cdot \frac{\partial o_t}{\partial V} = \frac{2}{N} \sum_{i=1}^{N} (y_t^i - \hat{y}_t^i) \cdot \sigma(o_t)(1 - \sigma(o_t)) \cdot h_t \tag{27}$$

BPTT Backward Pass

Backward Pass:

- $$\frac{\partial L}{\partial V} = \sum_t \frac{\partial L_t}{\partial V} \tag{24}$$
- $$\frac{\partial L_t}{\partial \hat{y}_t} = \frac{2}{N} \sum_{i=1}^{N} (y_t^i - \hat{y}_t^i) \tag{25}$$
- $$\frac{\partial L_t}{\partial o_t} = \frac{\partial L_t}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial o_t} = \frac{2}{N} \sum_{i=1}^{N} (y_t^i - \hat{y}_t^i) \cdot \sigma(o_t)(1 - \sigma(o_t)) \tag{26}$$
- $$\frac{\partial L_t}{\partial V} = \frac{\partial L_t}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial o_t} \cdot \frac{\partial o_t}{\partial V} = \frac{2}{N} \sum_{i=1}^{N} (y_t^i - \hat{y}_t^i) \cdot \sigma(o_t)(1 - \sigma(o_t)) \cdot h_t \tag{27}$$
- $$\frac{\partial L}{\partial V} = \sum_t \frac{\partial L_t}{\partial V} \tag{28}$$

BPTT Update

Update weights using Gradient Descent

BPTT Update

Update weights using Gradient Descent

- $$V = V^{old} - \lambda \cdot \frac{\partial L}{\partial V} \tag{29}$$

Contents

Related work

- Description of five state-of-the-art image-/video-prediction architectures

Related work

- Description of five state-of-the-art image-/video-prediction architectures
- Different approaches, but all use a LSTM as recurrent sub-module

Related work

- Description of five state-of-the-art image-/video-prediction architectures
- Different approaches, but all use a LSTM as recurrent sub-module
- Experiments are shown on MovingMNIST [7]

LSTM Autoencoder

- „Unsupervised Learning of Video Representations using LSTMs“by Srivastava et. al. [15]

LSTM Autoencoder

- „Unsupervised Learning of Video Representations using LSTMs“by Srivastava et. al. [15]
- Using the standard LSTM from Hochreiter & Schmidhuber [5]

LSTM Autoencoder

- „Unsupervised Learning of Video Representations using LSTMs“by Srivastava et. al. [15]
- Using the standard LSTM from Hochreiter & Schmidhuber [5]
- Autoencoder architecture

LSTM Autoencoder

- „Unsupervised Learning of Video Representations using LSTMs“by Srivastava et. al. [15]
- Using the standard LSTM from Hochreiter & Schmidhuber [5]
- Autoencoder architecture
- Used for future image prediction & image reconstruction

LSTM Autoencoder

- „Unsupervised Learning of Video Representations using LSTMs“by Srivastava et. al. [15]
- Using the standard LSTM from Hochreiter & Schmidhuber [5]
- Autoencoder architecture
- Used for future image prediction & image reconstruction
- Typical baseline for newer, more advanced algorithms

LSTM Autoencoder

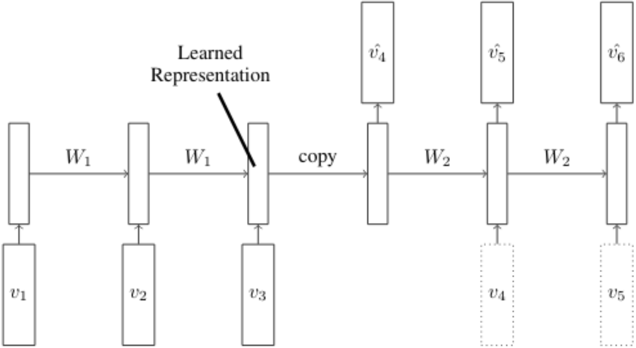


Figure: Future image prediction model [15]

LSTM Autoencoder

- Composed model can do image prediction and reconstruction simultaneously

LSTM Autoencoder

- Composed model can do image prediction and reconstruction simultaneously
    - Consisting of one encoder and two decoder

LSTM Autoencoder

- Composed model can do image prediction and reconstruction simultaneously
  - Consisting of one encoder and two decoder
  - Good example of representation learning algorithms

LSTM Autoencoder

- Composed model can do image prediction and reconstruction simultaneously
  - Consisting of one encoder and two decoder
  - Good example of representation learning algorithms
  - Both use the same encoder but differently trained decoder

LSTM Autoencoder

- Composed model can do image prediction and reconstruction simultaneously
  - Consisting of one encoder and two decoder
  - Good example of representation learning algorithms
  - Both use the same encoder but differently trained decoder
- End-to-End differentiable

LSTM Autoencoder

- Composed model can do image prediction and reconstruction simultaneously
  - Consisting of one encoder and two decoder
  - Good example of representation learning algorithms
  - Both use the same encoder but differently trained decoder
- End-to-End differentiable
- Trained with BPTT and CE-Loss for MovingMNIST (two digits per image)

LSTM Autoencoder

- Composed model can do image prediction and reconstruction simultaneously
  - Consisting of one encoder and two decoder
  - Good example of representation learning algorithms
  - Both use the same encoder but differently trained decoder
- End-to-End differentiable
- Trained with BPTT and CE-Loss for MovingMNIST (two digits per image)
- Used for multi-frame prediction 10 input images $\Rightarrow$ 10 output images

LSTM Autoencoder

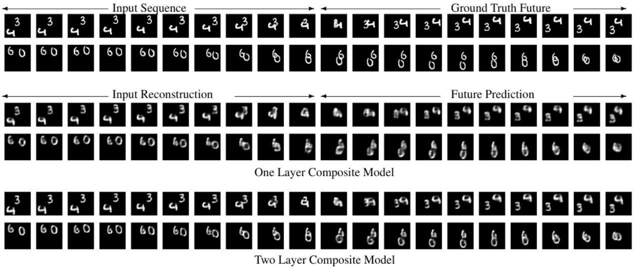


Figure: Results of MovingMNIST experiment [15]

ConvLSTM Autoencoder

- „Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting“by Shi et. al. [14]

ConvLSTM Autoencoder

- „Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting“by Shi et. al. [14]
- Similar to LSTM Autoencoder, but uses novel ConvLSTM instead

ConvLSTM Autoencoder

- „Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting“by Shi et. al. [14]
- Similar to LSTM Autoencoder, but uses novel ConvLSTM instead
- ConvLSTM with peephole connection

ConvLSTM Autoencoder

- „Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting“by Shi et. al. [14]
- Similar to LSTM Autoencoder, but uses novel ConvLSTM instead
- ConvLSTM with peephole connection
- Outperforms the LSTM Autoencoder „Captures spatiotemporal correlations better“

ConvLSTM Autoencoder

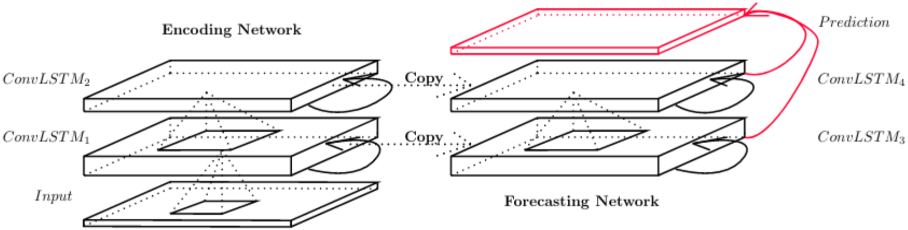


Figure: Future image prediction model [14]

ConvLSTM Autoencoder

- End-to-End differentiable

ConvLSTM Autoencoder

- End-to-End differentiable
- Trained using BPTT with CE-Loss for MovingMNIST (Three digits per image)

ConvLSTM Autoencoder

- End-to-End differentiable
- Trained using BPTT with CE-Loss for MovingMNIST (Three digits per image)
- Used for multi-frame prediction 10 input images $\Rightarrow$ 10 output images

ConvLSTM Autoencoder

- End-to-End differentiable
- Trained using BPTT with CE-Loss for MovingMNIST (Three digits per image)
- Used for multi-frame prediction 10 input images $\Rightarrow$ 10 output images
- Architecture is only used for forecasting, but might be able to have the same composed model as Srivastava et al. solution

ConvLSTM Autoencoder

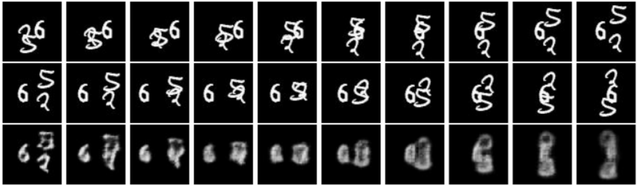


Figure: Results of MovingMNIST experiment [15]

Spatio-temporal Video Autoencoder

- „Spatio-Temporal Video Autoencoder With Differentiable Memory “by Patraucean et. al. [11]

Spatio-temporal Video Autoencoder

- „Spatio-Temporal Video Autoencoder With Differentiable Memory “by Patraucean et. al. [11]
- Using ConvLSTM as recurrent sub-module without peephole connection

Spatio-temporal Video Autoencoder

- „Spatio-Temporal Video Autoencoder With Differentiable Memory “by Patraucean et. al. [11]
- Using ConvLSTM as recurrent sub-module without peephole connection
- More complex architecture, by nesting an autoencoder inside another

Spatio-temporal Video Autoencoder

- „Spatio-Temporal Video Autoencoder With Differentiable Memory “by Patraucean et. al. [11]
- Using ConvLSTM as recurrent sub-module without peephole connection
- More complex architecture, by nesting an autoencoder inside another
    - The inner autoencoder is for temporal information

Spatio-temporal Video Autoencoder

- „Spatio-Temporal Video Autoencoder With Differentiable Memory “by Patraucean et. al. [11]
- Using ConvLSTM as recurrent sub-module without peephole connection
- More complex architecture, by nesting an autoencoder inside another
  - The inner autoencoder is for temporal information
  - The outer autoencoder is for spatial information

Spatio-temporal Video Autoencoder

- „Spatio-Temporal Video Autoencoder With Differentiable Memory “by Patraucean et. al. [11]
- Using ConvLSTM as recurrent sub-module without peephole connection
- More complex architecture, by nesting an autoencoder inside another
  - The inner autoencoder is for temporal information
  - The outer autoencoder is for spatial information
- Spatial autoencoder is a simple undercomplete autoencoder

Spatio-temporal Video Autoencoder

- „Spatio-Temporal Video Autoencoder With Differentiable Memory "by Patraucean et. al. [11]
- Using ConvLSTM as recurrent sub-module without peephole connection
- More complex architecture, by nesting an autoencoder inside another
  - The inner autoencoder is for temporal information
  - The outer autoencoder is for spatial information
- Spatial autoencoder is a simple undercomplete autoencoder
- Temporal autoencoder consists of a ConvLSTM as temporal encoder and optical flow as temporal decoder

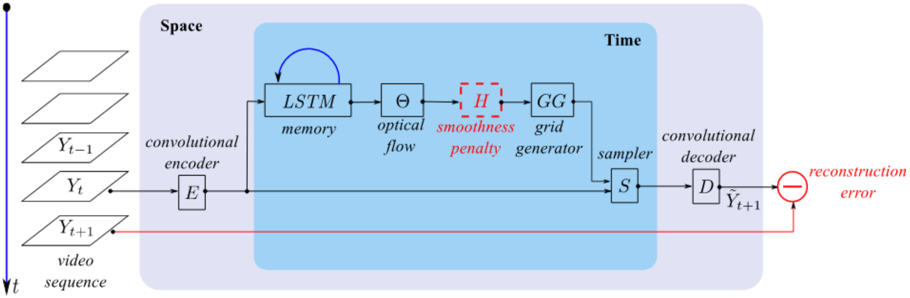


Figure: Spatio-temporal Video Autoencoder Architecture [11]

Spatio-temporal Video Autoencoder

- First input the whole image sequence X to the convolutional encoder (Works like a truncated SVD)

Spatio-temporal Video Autoencoder

- First input the whole image sequence X to the convolutional encoder (Works like a truncated SVD)
- Save the last output of the convolutional encoder

Spatio-temporal Video Autoencoder

- First input the whole image sequence X to the convolutional encoder (Works like a truncated SVD)
- Save the last output of the convolutional encoder
- Use the output sequence of the convolutional encoder as input for the ConvLSTM

Spatio-temporal Video Autoencoder

- First input the whole image sequence X to the convolutional encoder (Works like a truncated SVD)
- Save the last output of the convolutional encoder
- Use the output sequence of the convolutional encoder as input for the ConvLSTM
- Output of ConvLSTM will get into the optical flow convnet

Spatio-temporal Video Autoencoder

- First input the whole image sequence X to the convolutional encoder (Works like a truncated SVD)
- Save the last output of the convolutional encoder
- Use the output sequence of the convolutional encoder as input for the ConvLSTM
- Output of ConvLSTM will get into the optical flow convnet
- Normalize optical flow map and apply with saved image in image sampler

Spatio-temporal Video Autoencoder

- First input the whole image sequence X to the convolutional encoder (Works like a truncated SVD)
- Save the last output of the convolutional encoder
- Use the output sequence of the convolutional encoder as input for the ConvLSTM
- Output of ConvLSTM will get into the optical flow convnet
- Normalize optical flow map and apply with saved image in image sampler
- „Resize“next image.

Spatio-temporal Video Autoencoder

- End-to-End differentiable

Spatio-temporal Video Autoencoder

- End-to-End differentiable
- Trained using BPTT with CE-Loss for MovingMNIST (Two digits per image)

Spatio-temporal Video Autoencoder

- End-to-End differentiable
- Trained using BPTT with CE-Loss for MovingMNIST (Two digits per image)
- Used for one-frame prediction (19 images as input, one as output)

Spatio-temporal Video Autoencoder

- End-to-End differentiable
- Trained using BPTT with CE-Loss for MovingMNIST (Two digits per image)
- Used for one-frame prediction (19 images as input, one as output)
- Architecture could be used for multi-frame prediction with a feedback loop

Spatio-temporal Video Autoencoder

Figure: Results of MovingMNIST experiment [11]

PredNet

- „Deep Predictive Coding Networks For Video Prediction And Unsupervised Learning“by Lotter et al. [8]

PredNet

- „Deep Predictive Coding Networks For Video Prediction And Unsupervised Learning“by Lotter et al. [8]
- Informally named PredNet

PredNet

- „Deep Predictive Coding Networks For Video Prediction And Unsupervised Learning“by Lotter et al. [8]
- Informally named PredNet
- Architecture is based on the concept of „predictive coding“.

PredNet

- „Deep Predictive Coding Networks For Video Prediction And Unsupervised Learning“by Lotter et al. [8]
- Informally named PredNet
- Architecture is based on the concept of „predictive coding“.
  - Neuro-scientific task of „continually making predictions of incoming sensory stimuli“.

PredNet

- „Deep Predictive Coding Networks For Video Prediction And Unsupervised Learning“by Lotter et al. [8]
- Informally named PredNet
- Architecture is based on the concept of „predictive coding“.
  - Neuro-scientific task of „continually making predictions of incoming sensory stimuli“.
- Network consists of arbitrary amount of layers

PredNet

- „Deep Predictive Coding Networks For Video Prediction And Unsupervised Learning“by Lotter et al. [8]
- Informally named PredNet
- Architecture is based on the concept of „predictive coding“.
  - Neuro-scientific task of „continually making predictions of incoming sensory stimuli“.
- Network consists of arbitrary amount of layers
- Depth of network is hyperparameter

PredNet

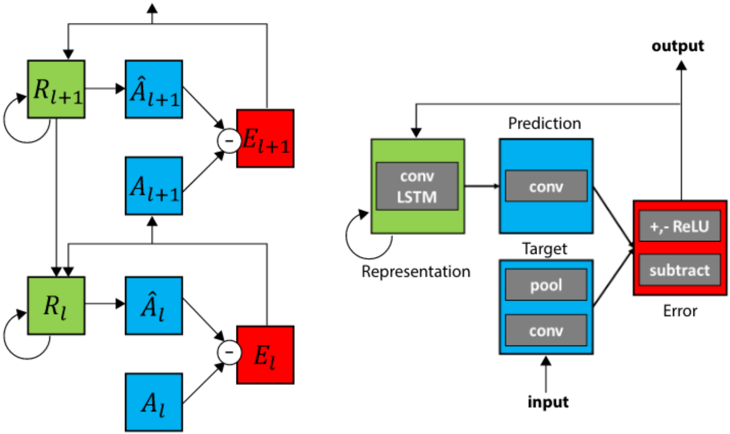


Figure: PredNet Architecture [8]

PredNet

- $$A_l{}^t = \begin{cases} x_t & l = 0 \\ MaxPool(ReLU(Conv(E_{l-1}^t))) & l > 0 \end{cases} \tag{30}$$

PredNet

- $$A_l{}^t = \begin{cases} x_t & l = 0 \\ MaxPool(ReLU(Conv(E_{l-1}^t))) & l > 0 \end{cases} \tag{30}$$

- $$\hat{A}_l^t = ReLU(Conv(R_l^t) \tag{31}$$

PredNet

- $$A_l{}^t = \begin{cases} x_t & l = 0 \\ MaxPool(ReLU(Conv(E_{l-1}^t))) & l > 0 \end{cases} \tag{30}$$

- $$\hat{A}_l^t = ReLU(Conv(R_l^t)) \tag{31}$$

- $$E_l^t = [ReLU(\hat{A}_l^t - A_l^t); ReLU(A_l^t - \hat{A}_l^t)] \tag{32}$$

PredNet

- $$A_l{}^t = \begin{cases} x_t & l = 0 \\ MaxPool(ReLU(Conv(E^t_{l-1}))) & l > 0 \end{cases} \qquad (30)$$

- $$\hat{A}^t_l = ReLU(Conv(R^t_l) \qquad (31)$$

- $$E^t_l = [ReLU(\hat{A}^t_l - A^t_l); ReLU(A^t_l - \hat{A}^t_l)] \qquad (32)$$

- $$R^t_l = ConvLSTM(E^{t-1}_l, R^{t-1}_l, Upsample(R^t_{l+1}) \qquad (33)$$

PredNet

- Doing prediction in every time-step instead after whole sequence

PredNet

- Doing prediction in every time-step instead after whole sequence
- First time-step will output a spatially uniform prediction

PredNet

- Doing prediction in every time-step instead after whole sequence
- First time-step will output a spatially uniform prediction
  - Because R_l and E_l are initialized with zero

PredNet

- Doing prediction in every time-step instead after whole sequence
- First time-step will output a spatially uniform prediction
  - Because R_l and E_l are initialized with zero
- First compute R_l^t states in the top-down pass

PredNet

- Doing prediction in every time-step instead after whole sequence
- First time-step will output a spatially uniform prediction
  - Because R_l and E_l are initialized with zero
- First compute R_l^t states in the top-down pass
- Then compute $\hat{A}_l^t$ in forward pass

PredNet

- Doing prediction in every time-step instead after whole sequence
- First time-step will output a spatially uniform prediction
  - Because R_l and E_l are initialized with zero
- First compute R_l^t states in the top-down pass
- Then compute $\hat{A}_l^t$ in forward pass
- If current layer is *neq* lowest layer, compute A_l^t

PredNet

- Doing prediction in every time-step instead after whole sequence
- First time-step will output a spatially uniform prediction
  - Because R_l and E_l are initialized with zero
- First compute R_l^t states in the top-down pass
- Then compute $\hat{A}_l^t$ in forward pass
- If current layer is *neq* lowest layer, compute A_l^t
  - A_0^t is an image of the input sequence

PredNet

- End-to-End differentiable

PredNet

- End-to-End differentiable
- Trained using BPTT

PredNet

- End-to-End differentiable
- Trained using BPTT
- Using an in-build error function

PredNet

- End-to-End differentiable
- Trained using BPTT
- Using an in-build error function
  - Network consists of two modes (Error and Prediction mode)

PredNet

- End-to-End differentiable
- Trained using BPTT
- Using an in-build error function
  - Network consists of two modes (Error and Prediction mode)
  - $$L_t = \sum_t \lambda_t \sum_l \frac{\lambda_l}{n_l} \sum_{n_l} E_l^t \tag{34}$$

PredNet

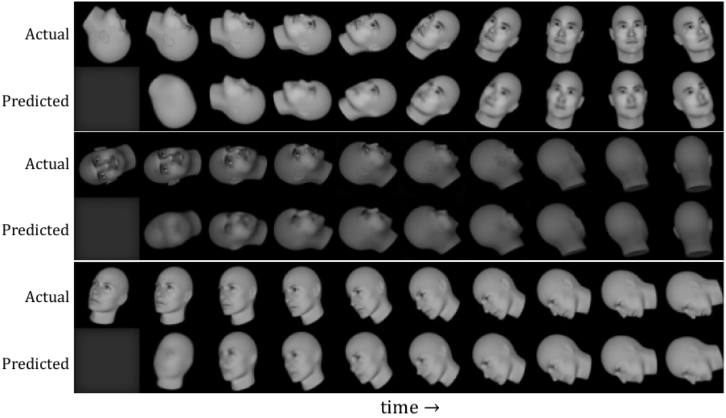
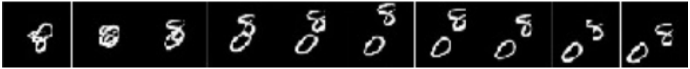


Figure: Results of FaceGen [8]

PredNet



(a) Ground truth

(b) ConvLSTM

Figure: Results for MovingMNIST [3].

PredRNN

- „PredRNN: Recurrent Neural Networks for Predictive Learning using Spatiotemporal LSTMs“by Wang et al. [16]

PredRNN

- „PredRNN: Recurrent Neural Networks for Predictive Learning using Spatiotemporal LSTMs“by Wang et al. [16]
- Novel LSTM module named Spatiotemporal LSTM (ST-LSTM)

PredRNN

- „PredRNN: Recurrent Neural Networks for Predictive Learning using Spatiotemporal LSTMs“by Wang et al. [16]
- Novel LSTM module named Spatiotemporal LSTM (ST-LSTM)
  - Consists of the typical temporal memory and extends this with an additional spatiotemporal memory

PredRNN

- „PredRNN: Recurrent Neural Networks for Predictive Learning using Spatiotemporal LSTMs“by Wang et al. [16]
- Novel LSTM module named Spatiotemporal LSTM (ST-LSTM)
  - Consists of the typical temporal memory and extends this with an additional spatiotemporal memory
- Novel Architecture named PredRNN

PredRNN

$$g_t = tanh(W_{xg} * X_t + W_{hg} * H^l_{t-1} + b_g) \tag{35}$$

$$i_t = \sigma(W_{xi} * X_t + W_{hi} * H^l_{t-1} + b_i) \tag{36}$$

$$f_t = \sigma(W_{xf} * X_t + W_{hf} * H^l_{t-1} + b_f) \tag{37}$$

$$C^l_t = f_t \odot C^l_{t-1} + i_t \odot g_t \tag{38}$$

$$g_{t\prime} = tanh(W_{xg\prime} * X_t + W_{mg} * M^{l-1}_t + b_{g\prime}) \tag{39}$$

$$i_{t\prime} = \sigma(W_{xi\prime} * X_t + W_{mi} * M^{l-1}_t + b_{i\prime}) \tag{40}$$

$$f_{t\prime} = \sigma(W_{xf\prime} * X_t + W_{mf} * M^{l-1}_t + b_{f\prime}) \tag{41}$$

$$M^l_t = f_{t\prime} \odot M^{l-1}_t + i_{t\prime} \odot g_{t\prime} \tag{42}$$

$$o_t = \sigma(W_{xo} * X_t + W_{ho} * H^l_{t-1} + W_{co} * C^l_t + W_{mo} * M^l_t + b_o) \tag{43}$$

$$H^l_t = o_t \odot tanh(W_{1\times 1} * [C^l_t, M^l_t]) \tag{44}$$

PredRNN

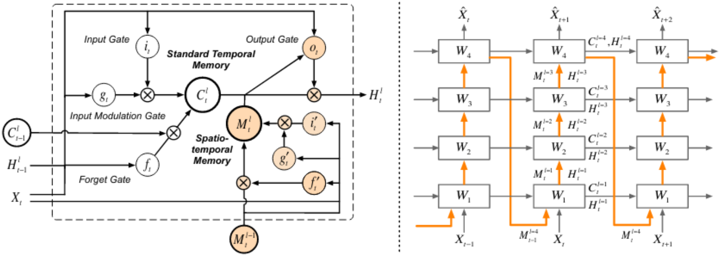


Figure: **Left:** ST-LSTM architecture, **Right:** PredRNN architecture [16]

PredRNN

- End-to-End differentiable

PredRNN

- End-to-End differentiable
- Trained using BPTT using CE-Loss on MovingMNIST (Two digits per frame)

PredRNN

- End-to-End differentiable
- Trained using BPTT using CE-Loss on MovingMNIST (Two digits per frame)
- Using ten images as input and output ten images

PredRNN

Figure: Results for MovingMNIST [16].

Related work

- Architectures have different approach but similar result

Related work

- Architectures have different approach but similar result
- First architecture uses LSTM in autoencoder architecture

Related work

- Architectures have different approach but similar result
- First architecture uses LSTM in autoencoder architecture
- The second architecture replaces the LSTM with ConvLSTM

Related work

- Architectures have different approach but similar result
- First architecture uses LSTM in autoencoder architecture
- The second architecture replaces the LSTM with ConvLSTM
- Third architecture leverages optical flow with two autoencoder, two split information in space and time

Related work

- Architectures have different approach but similar result
- First architecture uses LSTM in autoencoder architecture
- The second architecture replaces the LSTM with ConvLSTM
- Third architecture leverages optical flow with two autoencoder, two split information in space and time
- PredNet introduces a neuro-scientific idea to achieve state-of-the-art image-/video-prediction

Related work

- Architectures have different approach but similar result
- First architecture uses LSTM in autoencoder architecture
- The second architecture replaces the LSTM with ConvLSTM
- Third architecture leverages optical flow with two autoencoder, two split information in space and time
- PredNet introduces a neuro-scientific idea to achieve state-of-the-art image-/video-prediction
- The last architecture introduces a novel ConvLSTM named ST-LSTM and also a novel architecture named PredRNN

Related work

- Architectures have different approach but similar result
- First architecture uses LSTM in autoencoder architecture
- The second architecture replaces the LSTM with ConvLSTM
- Third architecture leverages optical flow with two autoencoder, two split information in space and time
- PredNet introduces a neuro-scientific idea to achieve state-of-the-art image-/video-prediction
- The last architecture introduces a novel ConvLSTM named ST-LSTM and also a novel architecture named PredRNN
- $\Rightarrow$ Even though, the algorithms have different approaches to perform state-of-the-art image-/video-prediction, they all use some kind of LSTM as recurrent module.

Contents

Experimental setup

- Use the three implemented networks and change the recurrent layer with PredRNN to see if this achieves better performance

Experimental setup

- Use the three implemented networks and change the recurrent layer with PredRNN to see if this achieves better performance
- Experiments were performed on three different datasets

Experimental setup

- Use the three implemented networks and change the recurrent layer with PredRNN to see if this achieves better performance
- Experiments were performed on three different datasets
  - MovingMNIST (synthetic dataset)

Experimental setup

- Use the three implemented networks and change the recurrent layer with PredRNN to see if this achieves better performance
- Experiments were performed on three different datasets
  - MovingMNIST (synthetic dataset)
    - Pre-processed to have frame size $(1 \times 64 \times 64)$, two digits per frame

Experimental setup

- Use the three implemented networks and change the recurrent layer with PredRNN to see if this achieves better performance
- Experiments were performed on three different datasets
  - MovingMNIST (synthetic dataset)
    - Pre-processed to have frame size $(1 \times 64 \times 64)$, two digits per frame
    - Training set of 10000 sequences, with every sequence consisting of ten images

Experimental setup

- Use the three implemented networks and change the recurrent layer with PredRNN to see if this achieves better performance
- Experiments were performed on three different datasets
  - MovingMNIST (synthetic dataset)
    - Pre-processed to have frame size $(1 \times 64 \times 64)$, two digits per frame
    - Training set of 10000 sequences, with every sequence consisting of ten images
    - Test set of 3000 sequences

Experimental setup

- Use the three implemented networks and change the recurrent layer with PredRNN to see if this achieves better performance
- Experiments were performed on three different datasets
    - MovingMNIST (synthetic dataset)
        - Pre-processed to have frame size ($1 \times 64 \times 64$), two digits per frame
        - Training set of 10000 sequences, with every sequence consisting of ten images
        - Test set of 3000 sequences
    - KTH (real dataset, fixed camera)

Experimental setup

- Use the three implemented networks and change the recurrent layer with PredRNN to see if this achieves better performance
- Experiments were performed on three different datasets
  - MovingMNIST (synthetic dataset)
    - Pre-processed to have frame size $(1 \times 64 \times 64)$, two digits per frame
    - Training set of 10000 sequences, with every sequence consisting of ten images
    - Test set of 3000 sequences
  - KTH (real dataset, fixed camera)
    - Pre-processed to gray-scaled images, cropped to size $(1 \times 80 \times 60)$

Experimental setup

- Use the three implemented networks and change the recurrent layer with PredRNN to see if this achieves better performance
- Experiments were performed on three different datasets
  - MovingMNIST (synthetic dataset)
    - Pre-processed to have frame size $(1 \times 64 \times 64)$, two digits per frame
    - Training set of 10000 sequences, with every sequence consisting of ten images
    - Test set of 3000 sequences
  - KTH (real dataset, fixed camera)
    - Pre-processed to gray-scaled images, cropped to size $(1 \times 80 \times 60)$
  - Kitti (real dataset, ego-centric movement)

Experimental setup

- Use the three implemented networks and change the recurrent layer with PredRNN to see if this achieves better performance
- Experiments were performed on three different datasets
  - MovingMNIST (synthetic dataset)
    - Pre-processed to have frame size $(1 \times 64 \times 64)$, two digits per frame
    - Training set of 10000 sequences, with every sequence consisting of ten images
    - Test set of 3000 sequences
  - KTH (real dataset, fixed camera)
    - Pre-processed to gray-scaled images, cropped to size $(1 \times 80 \times 60)$
  - Kitti (real dataset, ego-centric movement)
    - Cropped to frame size $(3 \times 160 \times 128)$

Experimental setup

- First experiments were performed using fixed hyperparameter (HP optimization is only applied in the second experiments)

Experimental setup

- First experiments were performed using fixed hyperparameter (HP optimization is only applied in the second experiments)
  - In total 18 different networks trained

Experimental setup

- First experiments were performed using fixed hyperparameter (HP optimization is only applied in the second experiments)
  - In total 18 different networks trained
  - On own Computer $\Rightarrow$ Kitti training took > 12h

Experimental setup

- First experiments were performed using fixed hyperparameter (HP optimization is only applied in the second experiments)
  - In total 18 different networks trained
  - On own Computer $\Rightarrow$ Kitti training took > 12h
- MovingMNIST 5000 iterations per epoch in one epoch (Inspired by Elsayed et al. [3])

Experimental setup

- First experiments were performed using fixed hyperparameter (HP optimization is only applied in the second experiments)
  - In total 18 different networks trained
  - On own Computer $\Rightarrow$ Kitti training took > 12h
- MovingMNIST 5000 iterations per epoch in one epoch (Inspired by Elsayed et al. [3])
- KTH 500 iterations per epoch in 20 epochs

Experimental setup

- First experiments were performed using fixed hyperparameter (HP optimization is only applied in the second experiments)
  - In total 18 different networks trained
  - On own Computer $\Rightarrow$ Kitti training took $> 12h$
- MovingMNIST 5000 iterations per epoch in one epoch (Inspired by Elsayed et al. [3])
- KTH 500 iterations per epoch in 20 epochs
- Kitti 500 iterations per epoch in 50 epochs

Experimental setup

- First experiments were performed using fixed hyperparameter (HP optimization is only applied in the second experiments)
  - In total 18 different networks trained
  - On own Computer $\Rightarrow$ Kitti training took > 12h
- MovingMNIST 5000 iterations per epoch in one epoch (Inspired by Elsayed et al. [3])
- KTH 500 iterations per epoch in 20 epochs
- Kitti 500 iterations per epoch in 50 epochs
- Adam [6] was used as optimizer with lr of 0.001 and lr-scheduler reducing the lr after $\frac{epochs}{2}$ by factor ten

Experimental setup

- First experiments were performed using fixed hyperparameter (HP optimization is only applied in the second experiments)
  - In total 18 different networks trained
  - On own Computer $\Rightarrow$ Kitti training took $>$ 12h
- MovingMNIST 5000 iterations per epoch in one epoch (Inspired by Elsayed et al. [3])
- KTH 500 iterations per epoch in 20 epochs
- Kitti 500 iterations per epoch in 50 epochs
- Adam [6] was used as optimizer with lr of 0.001 and lr-scheduler reducing the lr after $\frac{epochs}{2}$ by factor ten
- Other hyperparamter can be found in the yml-folder of the implementation

Experimental setup

- Second experiments are performed using HP optimization, using early-stopping and with „optimal values“

Experimental setup

- Second experiments are performed using HP optimization, using early-stopping and with „optimal values“
- Only one out of 18 experiments performed, to get the importance of HP optimization and a tendency of how the results should look using optimal conditions

Experimental results (First setup)

- Section is splitted into the three different datasets

Experimental results (First setup)

- Section is splitted into the three different datasets
- First comparison of trainable parameter amount

Experimental results (First setup)

- Section is splitted into the three different datasets
- First comparison of trainable parameter amount
- Secondly table of mean MSE for testing and some example output

MovingMNIST

Model	ConvLSTM	PredRNN
Autoencoder (Depth 2)	537.411	773.018
PredNet	6.909.818	12.421.090
Spatiotemp	1.001.324	1.415.639

Table: Number of trainable parameter for MovingMNIST.

MovingMNIST

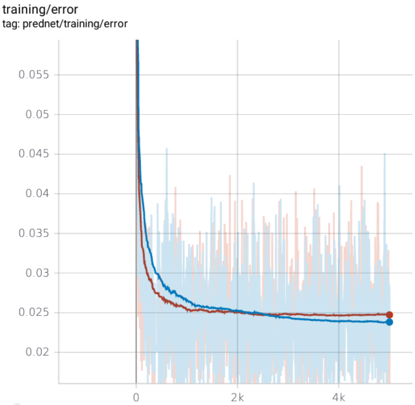
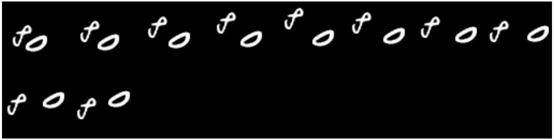


Figure: Training PredNet with ConvLSTM (blue) and PredRNN (red) on MovingMNIST.

MovingMNIST



(a) Ground truth

(b) ConvLSTM

(c) PredRNN (vanished)

Figure: Test results for PredNet on MovingMNIST.

MovingMNIST

Model	ConvLSTM	PredRNN
Autoencoder (Depth 2)	0.027	0.028
PredNet	0.035	0.041
Spatiotemp	0.024	0.022

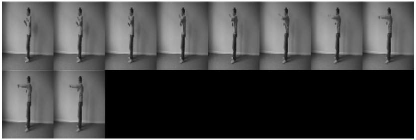
Table: Mean MSE for MovingMNIST.

KTH

Model	ConvLSTM	PredRNN
Autoencoder (Depth 2)	542.531	781.760
PredNet	850.325	1.285.730
Spatiotemp	1.007.598	1.421.913

Table: Number of trainable parameter for KTH.

KTH



(a) Ground truth

(b) ConvLSTM

(c) PredRNN

Figure: Test results for PredNet on KTH.

KTH

Model	**ConvLSTM**	**PredRNN**
Autoencoder (Depth 2)	$1.55e-3$	0.05 (didn't converged)
PredNet	$1.95e-3$	$1.93e-3$
Spatiotemp	$3.1e-3$	0.025 (didn't converged)

Table: Mean MSE for KTH.

Kitti

Model	ConvLSTM	PredRNN
Autoencoder (Depth 2)	542.531	781.760
PredNet	8.222.559	12.430.626
Spatiotemp	1.640.321	2.200.641

Table: Number of trainable parameter for Kitti.

Kitti

(a) Ground truth

(b) ConvLSTM

(c) PredRNN

Figure: Test results for PredNet on Kitti.

Kitti

Model	ConvLSTM	PredRNN
Autoencoder (Depth 2)	0.02	0.013
PredNet	0.019	0.02
Spatiotemp	0.018	0.017

Table: Mean MSE for Kitti.

Experimental Results (Second setup)

- Performed on KTH dataset with optimal values and lr of 0.0001

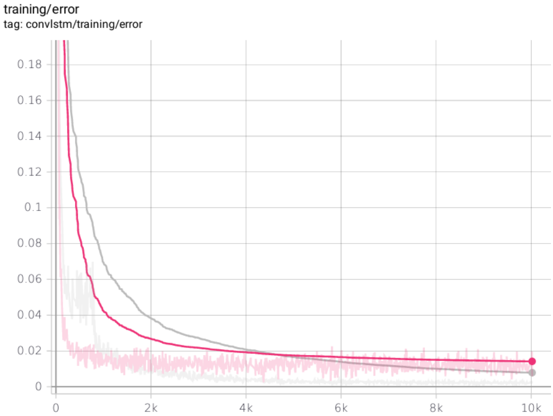


Figure: Training Autoencoder with ConvLSTM (red) and PredRNN (gray) on KTH using hyperparameter optimization and early stopping.

Experimental Results (Second setup)

- Mean test MSE for Autoencoder using ConvLSTM: 0.0013

Experimental Results (Second setup)

- Mean test MSE for Autoencoder using ConvLSTM: 0.0013
- Mean test MSE for Autoencoder using PredRNN: 0.00072

Experimental Results (Second setup)

- Mean test MSE for Autoencoder using ConvLSTM: 0.0013
- Mean test MSE for Autoencoder using PredRNN: 0.00072
- PredRNN is able to boost Autoencoder by more then 80 percent

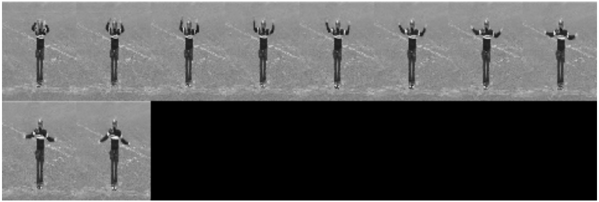
Experimental Results (Second setup)

- Mean test MSE for Autoencoder using ConvLSTM: 0.0013
- Mean test MSE for Autoencoder using PredRNN: 0.00072
- PredRNN is able to boost Autoencoder by more then 80 percent
- Showed importance of HP optimization and early stopping

Experimental Results (Second setup)

- Mean test MSE for Autoencoder using ConvLSTM: 0.0013
- Mean test MSE for Autoencoder using PredRNN: 0.00072
- PredRNN is able to boost Autoencoder by more then 80 percent
- Showed importance of HP optimization and early stopping
- Gives a hint of how the other experiments should perform

Experimental Results (Second setup)



(a) Ground truth

(b) ConvLSTM

(c) PredRNN

Figure: Test results for Autoencoder on KTH using hyperparameter optimization and early stopping.

Contents

Discussion

- Related work showed different state-of-the-art solutions for image-/video-prediction

Discussion

- Related work showed different state-of-the-art solutions for image-/video-prediction
  - Showed how previous work is used and adapted to gain better performance

Discussion

- Related work showed different state-of-the-art solutions for image-/video-prediction
  - Showed how previous work is used and adapted to gain better performance
  - E.g. Shi et al. [14] used the autoencoder architecture from Srivastava et al. [15] but with novel ConvLSTM module

Discussion

- Related work showed different state-of-the-art solutions for image-/video-prediction
  - Showed how previous work is used and adapted to gain better performance
  - E.g. Shi et al. [14] used the autoencoder architecture from Srivastava et al. [15] but with novel ConvLSTM module
  - All have a different approach to get to the same goal and rely on the same recurrent module basis

Discussion

- Related work showed different state-of-the-art solutions for image-/video-prediction
  - Showed how previous work is used and adapted to gain better performance
  - E.g. Shi et al. [14] used the autoencoder architecture from Srivastava et al. [15] but with novel ConvLSTM module
  - All have a different approach to get to the same goal and rely on the same recurrent module basis
- Network implementation showed, that a researcher should have basic skills in Python and more then one deep learning framework, in order to be able to read and understand code

Discussion

- Related work showed different state-of-the-art solutions for image-/video-prediction
  - Showed how previous work is used and adapted to gain better performance
  - E.g. Shi et al. [14] used the autoencoder architecture from Srivastava et al. [15] but with novel ConvLSTM module
  - All have a different approach to get to the same goal and rely on the same recurrent module basis
- Network implementation showed, that a researcher should have basic skills in Python and more then one deep learning framework, in order to be able to read and understand code
- First experiments were fixed and didn't showed any superior behavior of PredRNN as recurrent layer

Discussion

- Related work showed different state-of-the-art solutions for image-/video-prediction
  - Showed how previous work is used and adapted to gain better performance
  - E.g. Shi et al. [14] used the autoencoder architecture from Srivastava et al. [15] but with novel ConvLSTM module
  - All have a different approach to get to the same goal and rely on the same recurrent module basis
- Network implementation showed, that a researcher should have basic skills in Python and more then one deep learning framework, in order to be able to read and understand code
- First experiments were fixed and didn't showed any superior behavior of PredRNN as recurrent layer
- Using HP optimization, the tables turned and even one example outperformed the ConvLSTM solution with a performance boost of $> 80\%$

Contents

Conclusion

- Different approaches for image-/video-prediction were discussed

Conclusion

- Different approaches for image-/video-prediction were discussed
- The choice of the recurrent module is important for the performance of the network

Conclusion

- Different approaches for image-/video-prediction were discussed
- The choice of the recurrent module is important for the performance of the network
- The results showed, how important HP optimization is to achieve state-of-the-art performance and that without, having a more complex sub-module is not the key

Conclusion

- Different approaches for image-/video-prediction were discussed
- The choice of the recurrent module is important for the performance of the network
- The results showed, how important HP optimization is to achieve state-of-the-art performance and that without, having a more complex sub-module is not the key
- For future experiments, it would be nice to have all 18 experiments performed with HP optimization to get the overall results

Bibliography

[1] *CNN — Introduction to Pooling Layer*. https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/. – Accessed: 2020-08-27

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

[3] Elsayed, Nelly ; Maida, Anthony S. ; Bayoumi, Magdy A.: Reduced-Gate Convolutional LSTM Using Predictive Coding for Spatiotemporal Prediction. In: *CoRR* abs/1810.07251 (2018). – URL http://arxiv.org/abs/1810.07251

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

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

[6] Kingma, Diederik P. ; Ba, Jimmy: Adam: A Method for Stochastic Optimization. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, URL http://arxiv.org/abs/1412.6980, 2015

[7] LeCun, Y. ; Bottou, L. ; Bengio, Y. ; Haffner, P.: Gradient-based