

Lab CudaVision

Learning Vision Systems on Graphics Cards (MA-INF 4308)

Recurrent Neural Networks

09.12.2022

PROF. SVEN BEHNKE, ANGEL VILLAR-CORRALES

Contact: villar@ais.uni-bonn.de

Motivation

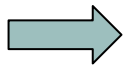
Why RNNs?

- Lots of **sequential** or **time-dependent** data:

- Speech and Audio
- Video
- Sensor data



- ‘Snapshots’ are often not informative



Integrate temporal context into the network

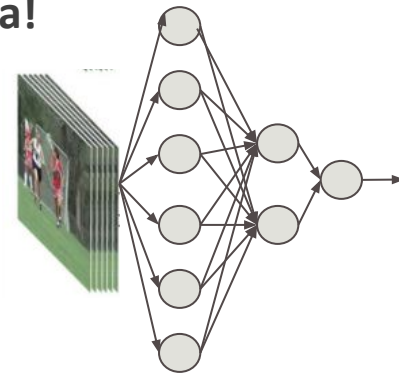
Temporal Context

- How can we integrate temporal context in the network?



1. Feed whole sequence to a big network: **(In General) Bad Idea!**

- Too many parameters
- Unclear how to distinguish spatial and time dimensions
- Cannot handle real time



2. Sliding window approaches

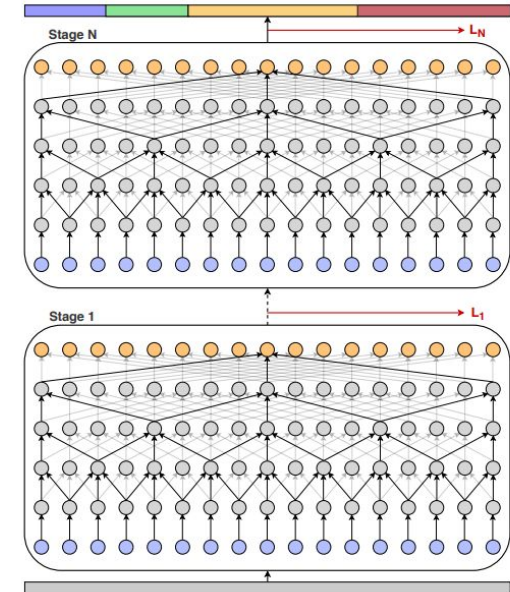
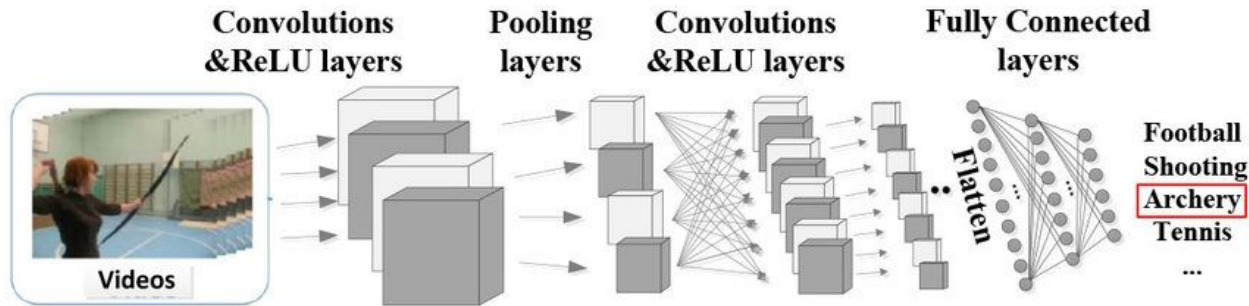
- Process a subset of the inputs at the time

3. Model sequential behavior within the architecture

- **Recurrent Neural Networks (RNNs)**

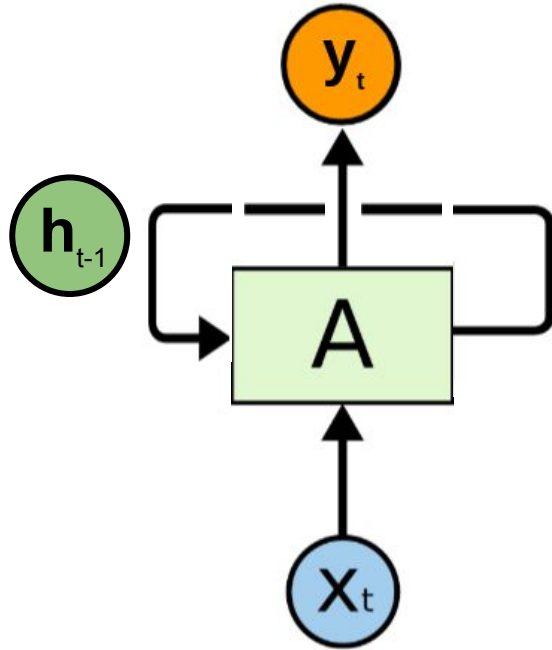
Disclaimer

- Powerful CNN-based architectures for video processing
 - 3D-CNNs
 - Temporal Convolutional Networks (TCNs)
- Better than RNNs on many video tasks



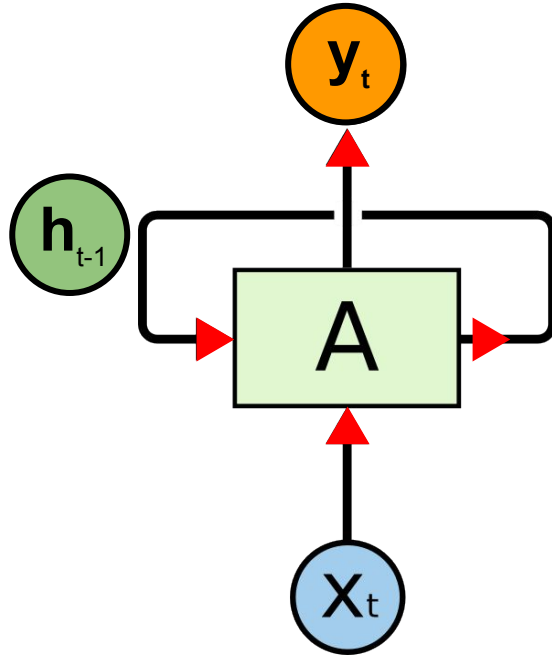
Simple RNNs

Simple RNNs



- RNNs include inner temporal loops
- Allow information to flow temporally
- Have been around since the 70s
 - For longer than CNNs!

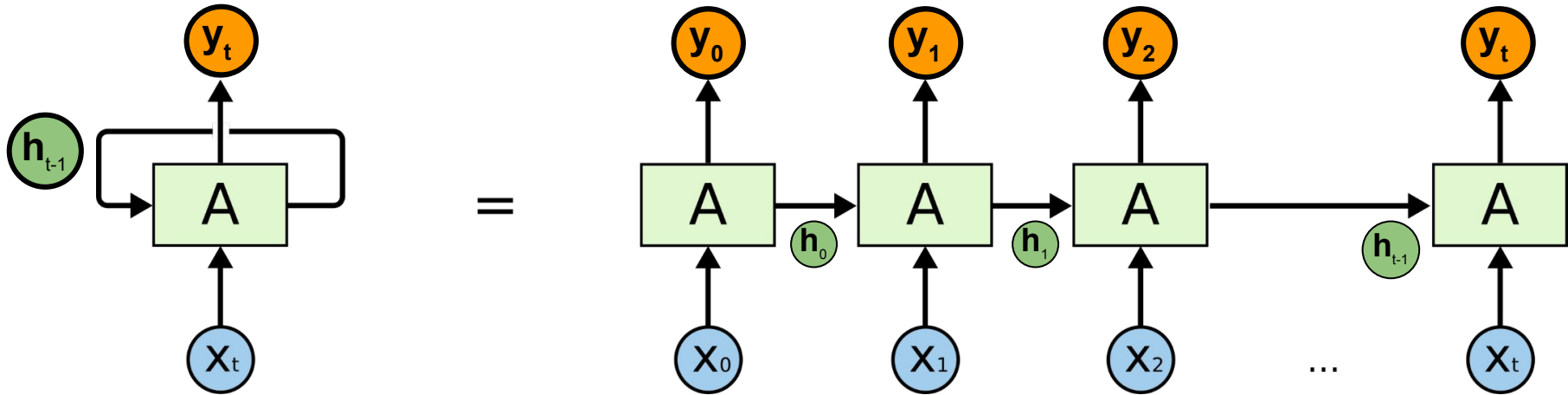
RNN Structure



- **Input** at time t : X_t is feed to a network
- **Extra input: Hidden state** h_{t-1} obtained by the unit
- **Outputs**
 - Y_t : output with information from present and past
 - h_t : next hidden state

RNN Structure

- ‘Unfolded RNN’: sequence of copies of the **same** RNN cell
 - Temporal weight sharing
 - Each unit passes a hidden state as input to its successor



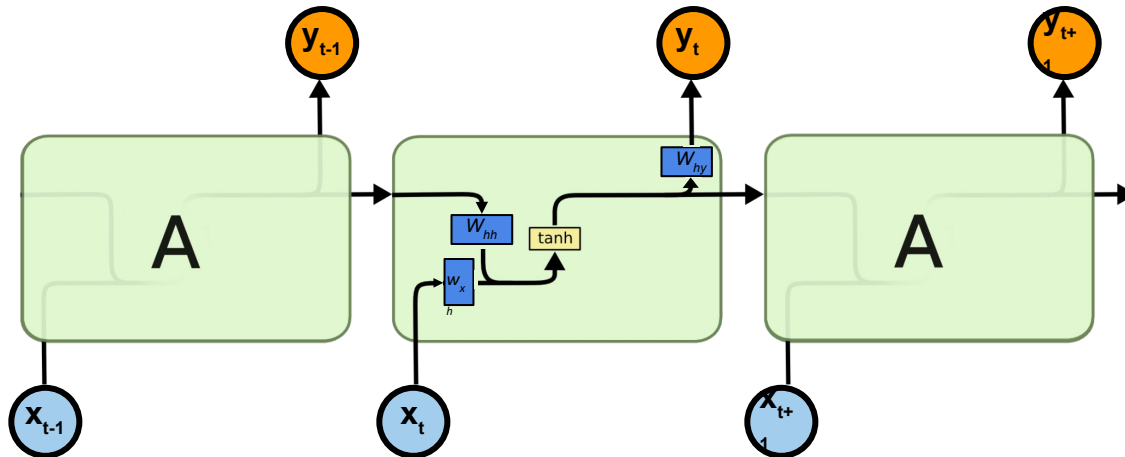
Vanilla RNN

- Next Hidden State

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

- Output at current time step

$$y_t = W_{hy}h_t$$



Why not using the ReLU function?

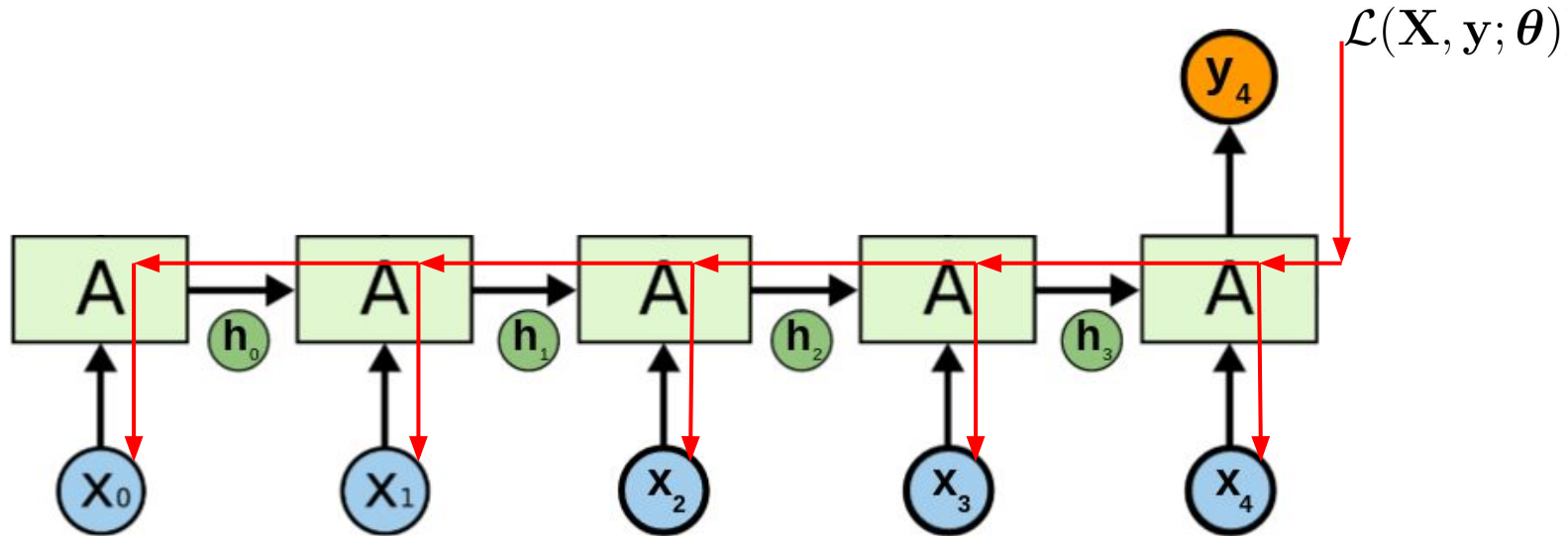
Vanilla RNN

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \quad y_t = W_{hy}h_t$$

```
class RNN:
    # ...
    def step(self, x):
        # update the hidden state
        self.h = np.tanh(np.dot(self.W_hh, self.h) + np.dot(self.W_xh, x))
        # compute the output vector
        y = np.dot(self.W_hy, self.h)
        return y
```

RNN Training

- RNNs are trained with backpropagation through time (BPTT)



Deep RNN

- We can stack several RNNs
 - Similar to CNNs
- Very common to use stack least two RNNs

Residual Stacked RNNs for Action Recognition

Mohamed Ilyes Lakkhal¹, Albert Clapés², Sergio Escalera², Oswald Lanz³, and Andrea Cavallaro¹

¹ CIS, Queen Mary University of London, UK
{m.i.lakkhal, a.cavallaro}@qmul.ac.uk

² Computer Vision Centre and University of Barcelona, Spain
{aclapes, sergio.escalera.guerrero}@gmail.com

³ TeV, Fondazione Bruno Kessler, Trento, Italy
lanz@fbk.eu

Deep RNNs Encode Soft Hierarchical Syntax

Terra Blevins, Omer Levy, and Luke Zettlemoyer
Paul G. Allen School of Computer Science & Engineering
University of Washington
Seattle, WA

Residual Stacking of RNNs for Neural Machine Translation

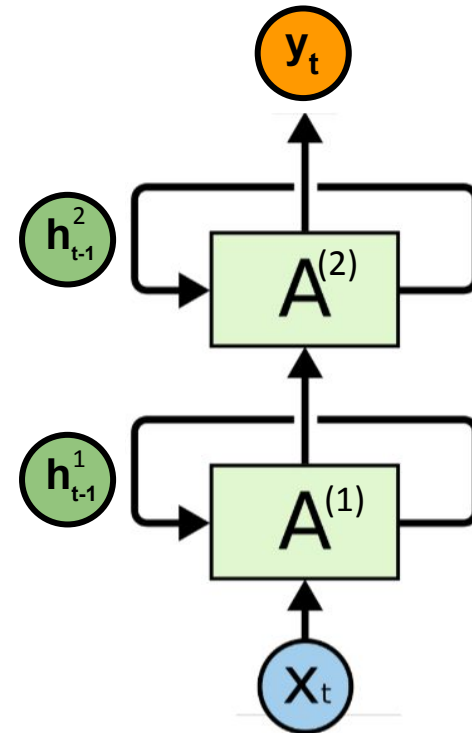
Raphael Shu Akiva Miura
The University of Tokyo Nara Institute of Science and Technology
shu@nlab.ci.i.u-tokyo.ac.jp miura.akiba.lr9@is.naist.jp

How to Construct Deep Recurrent Neural Networks

Razvan Pascanu¹, Caglar Gulcehre¹, Kyunghyun Cho², and Yoshua Bengio¹

¹Département d'Informatique et de Recherche Opérationnelle, Université de Montréal,
{pascanur, gulcehrc}@iro.umontreal.ca, yoshua.bengio@umontreal.ca

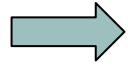
²Department of Information and Computer Science, Aalto University School of Science,
kyunghyun.cho@aalto.fi



Problem with RNNs

Gradient Problem

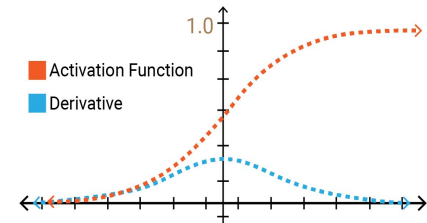
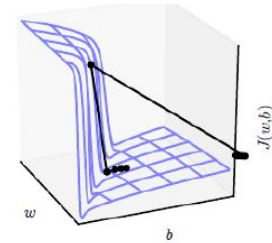
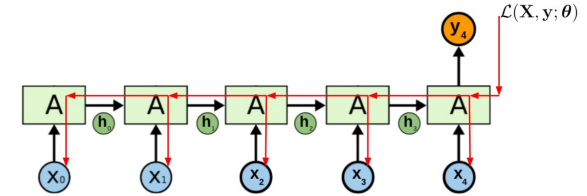
- Weights/Gradients are computed by a deep cascade of matrix multiplications and TanH nonlinearities



Gradients prone to vanishing or exploding

- Exploding gradient:** just clip the gradients

- Vanishing gradient:** hard to solve



Memory Overwriting

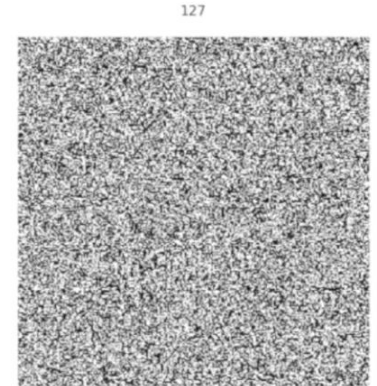
- RNNs have it difficult to connect past and present for long time spans
- Hidden state is overwritten every time step

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

Q: Can we do better?



A: Yes!



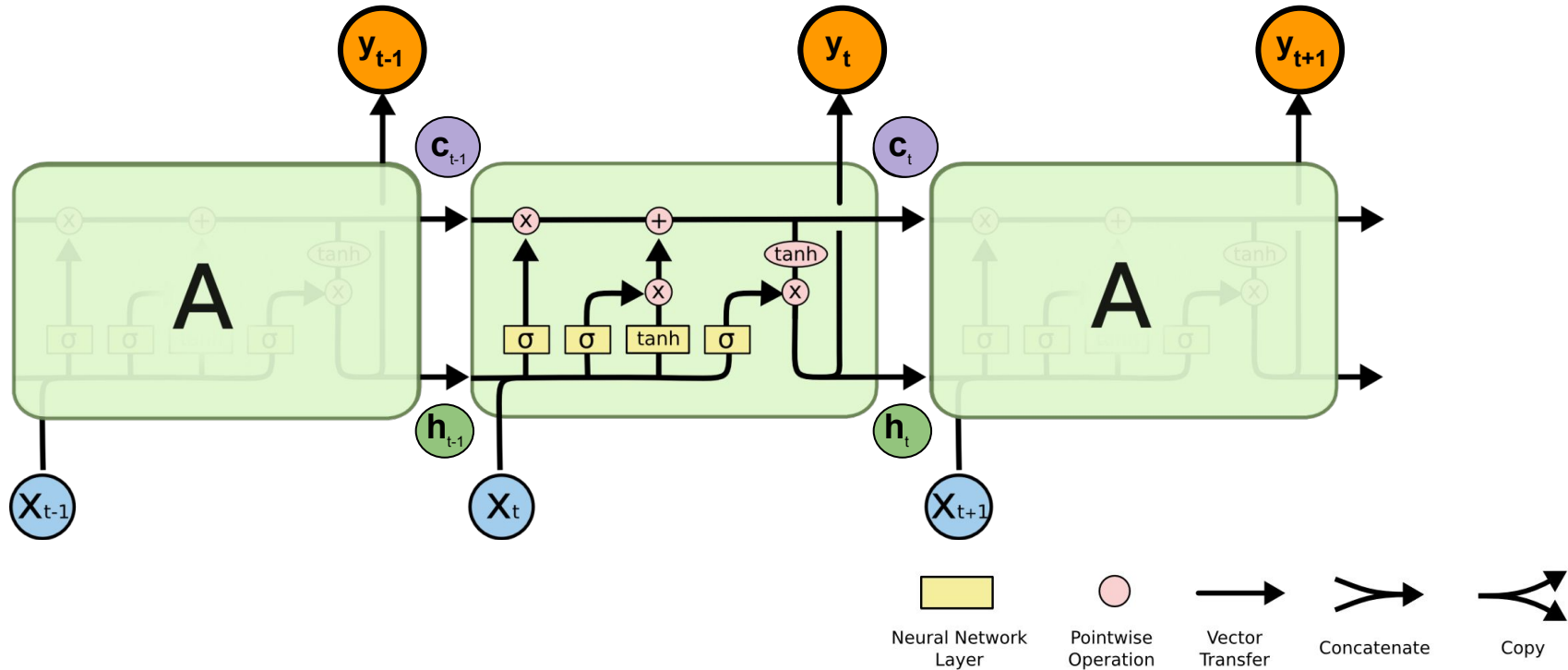
Long Short-Term Memory (LSTM)

Long Short-Term Memory

- Introduced by Hochreiter & Schmidhuber in 1997
- Designed to model long-term dependencies
- **Core idea:** using gates to control access and writing in an additional cell state

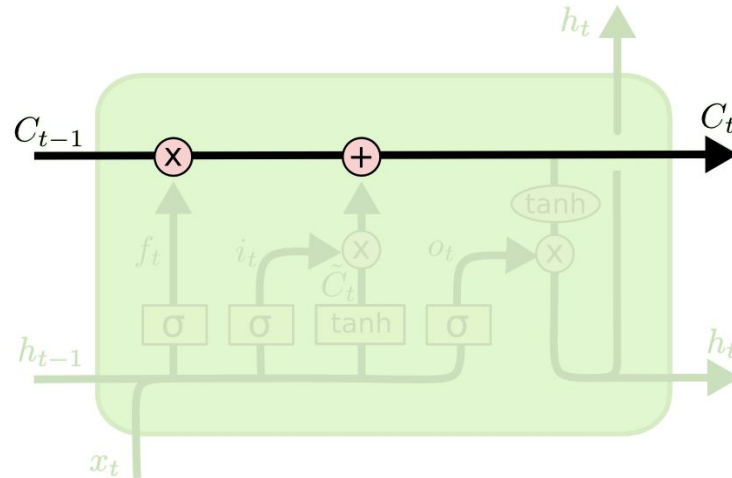


LSTM Structure



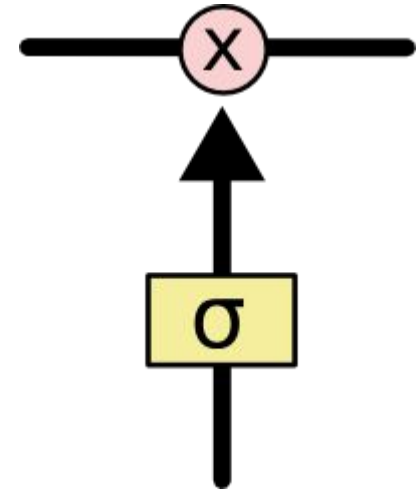
Cell State

- Runs across the entire LSTM with just some minor linear interaction
- LSTM modifies cell state using the so-called gates
- Cell state (C) \neq Hidden state (H)



Gates

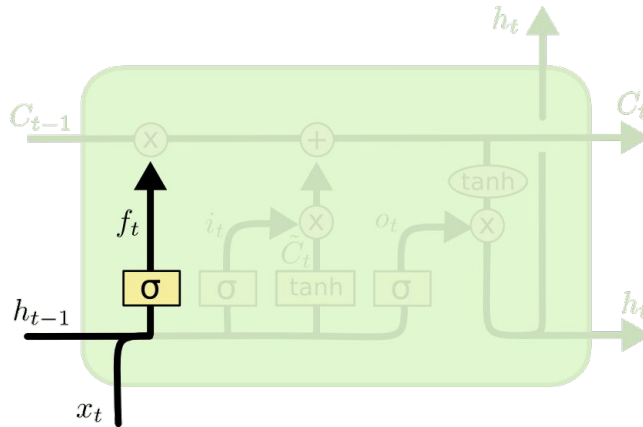
- **Gating** is a way of letting information through
- Gates are composed of a sigmoid activation and a pointwise product
- Sigmoid determines how much information goes through
 - **Zero:** Let nothing through
 - **One:** Let everything through



Step-by-Step LSTM Walk Through

Forget Gate

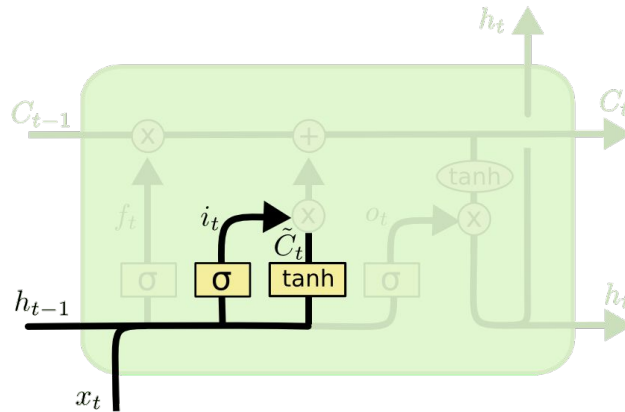
- Decides which information is removed from the cell state
- Input and hidden state determine which values in cell state are forgotten



$$f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f)$$

Input Gate

- Determines what new information goes into the cell state
- Two elements:
 - Sigmoid decides which values to update
 - TanH creates candidate values to update the cell state

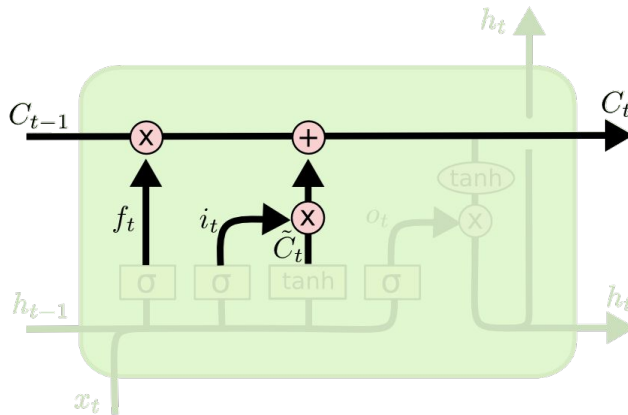


$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

Update Gate

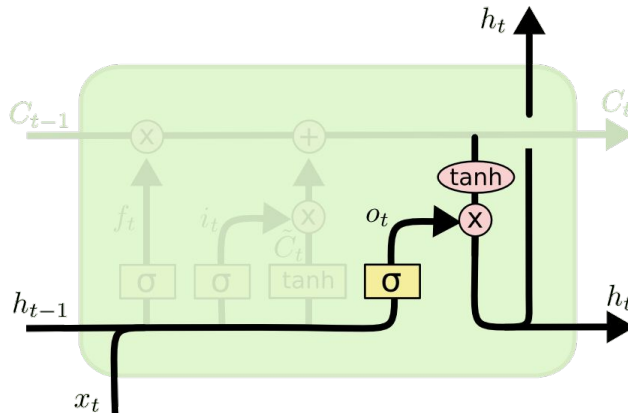
- Updates the old cell state into the new cell state
- Sequentially applies the results from the *forget* and *input* gates



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Output

- Forms the output based on a filtered version of the input and states
- Input and hidden state select what information of cell state goes to the output



$$o_t = \sigma (W_o [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh (C_t)$$

Summary

- Forget Gate

$$f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f)$$

- Input Gate

$$i_t = \sigma (W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

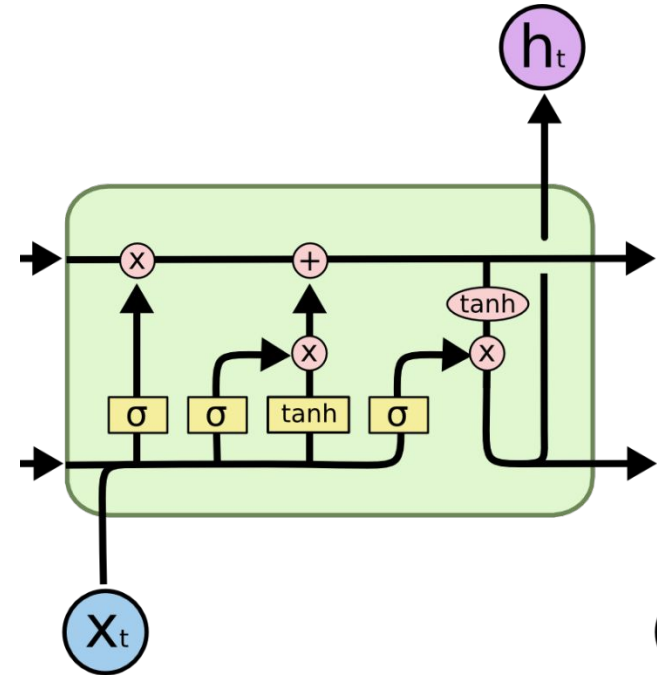
- Update Gate

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

- Output Gate

$$o_t = \sigma (W_o [h_{t-1}, x_t] + b_o)$$

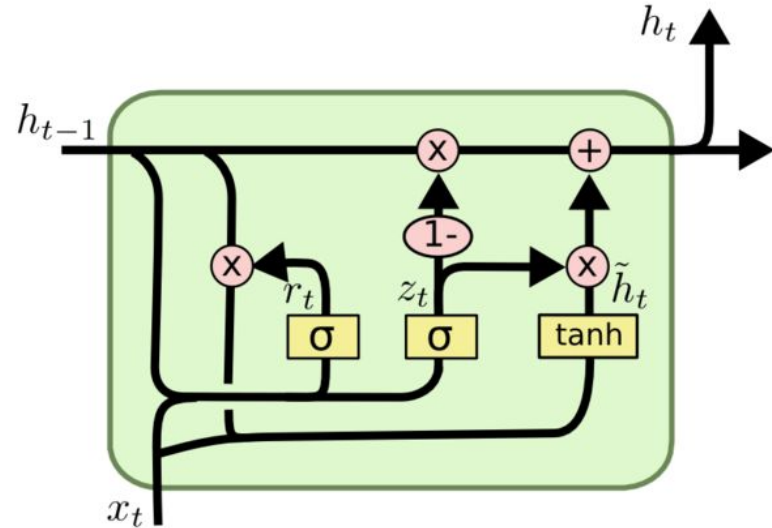
$$h_t = o_t * \tanh (C_t)$$



LSTM Variants

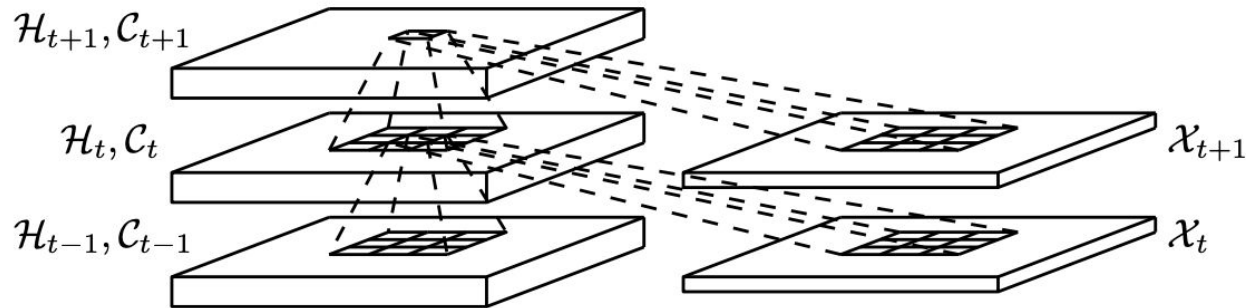
Gated Recurrent Unit

- Similar to LSTM, but 'simpler' and with fewer parameters
- Uses gates to control state
- Hidden and cell states are joined!



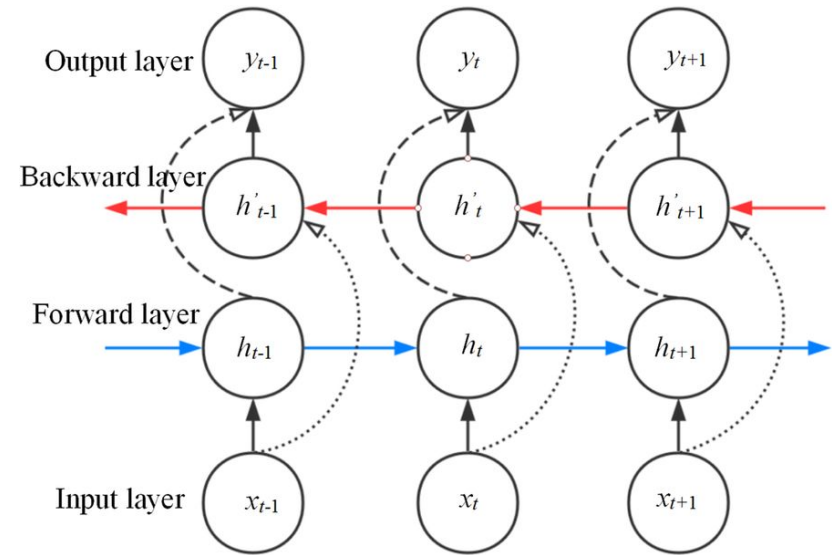
Conv-LSTM

- Spatio-temporal neural network
 - Combines ideas from CNN's and RNNs
- Uses conv. structures in both the input and state transitions



Bidirectional RNN

- Simultaneously look at past and future samples
- Uses two different states:
 - One in forward time direction
 - Other in backwards time direction
- Cannot handle real-time processing



Some Applications of RNNs

RNNs for Images

- RNNs can also be applied to datasets with fixed inputs, such as images

MULTIPLE OBJECT RECOGNITION WITH VISUAL ATTENTION

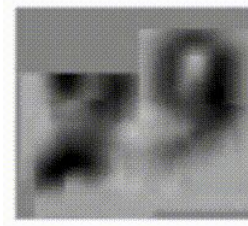
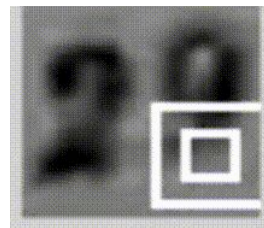
Jimmy Lei Ba*
University of Toronto
jimmy@psi.utoronto.ca

Volodymyr Mnih
Google DeepMind
vmnih@google.com

Koray Kavukcuoglu
Google DeepMind
korayk@google.com

ABSTRACT

We present an attention-based model for recognizing multiple objects in images. The proposed model is a deep recurrent neural network trained with reinforcement learning to attend to the most relevant regions of the input image. We show that the model learns to both localize and recognize multiple objects despite being given only class labels during training. We evaluate the model on the challenging task of transcribing house number sequences from Google Street View images and show that it is both more accurate than the state-of-the-art convolutional networks and uses fewer parameters and less computation.



RNNs for Images

- RNNs can also be applied to datasets with fixed inputs, such as images

DRAW: A Recurrent Neural Network For Image Generation

Karol Gregor
Ivo Danihelka
Alex Graves
Danilo Jimenez Rezende
Daan Wierstra
Google DeepMind

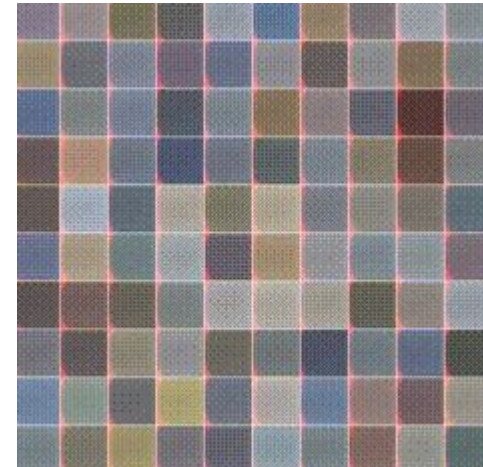
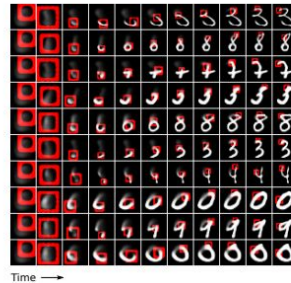
KAROLG@GOOGLE.COM
DANIHELKA@GOOGLE.COM
GRAVES@GOOGLE.COM
DANILOR@GOOGLE.COM
WIERSTRA@GOOGLE.COM

Abstract

This paper introduces the *Deep Recurrent Attentive Writer* (DRAW) neural network architecture for image generation. DRAW networks combine a novel spatial attention mechanism that mimics the foveation of the human eye, with a sequential variational auto-encoding framework that allows for the iterative construction of complex images. The system substantially improves on the state of the art for generative models on MNIST, and, when trained on the Street View House Numbers dataset, it generates images that cannot be distinguished from real data with the naked eye.

1. Introduction

A sequence of images generated by the DRAW network.



RNNs for Images

- RNNs can also be applied to datasets with fixed inputs, such as images

Pixel Recurrent Neural Networks

Aäron van den Oord
Nal Kalchbrenner
Koray Kavukcuoglu

Google DeepMind

AVDNOORD@GOOGLE.COM
NALK@GOOGLE.COM
KORAYK@GOOGLE.COM

Abstract

Modeling the distribution of natural images is a landmark problem in unsupervised learning. This task requires an image model that is at once expressive, tractable and scalable. We present a deep neural network that sequentially predicts the pixels in an image along the two spatial dimensions. Our method models the discrete probability of the raw pixel values and encodes the complete set of dependencies in the image. Architectural novelties include fast two-dimensional recurrent layers and an effective use of residual connections in deep recurrent networks. We achieve log-likelihood scores on natural images that are considerably better than the previous state of the art. Our main results also provide benchmarks on the diverse ImageNet dataset. Samples generated from the model appear crisp, varied and globally coherent.



Figure 1. Image completions sampled from a PixelRNN.

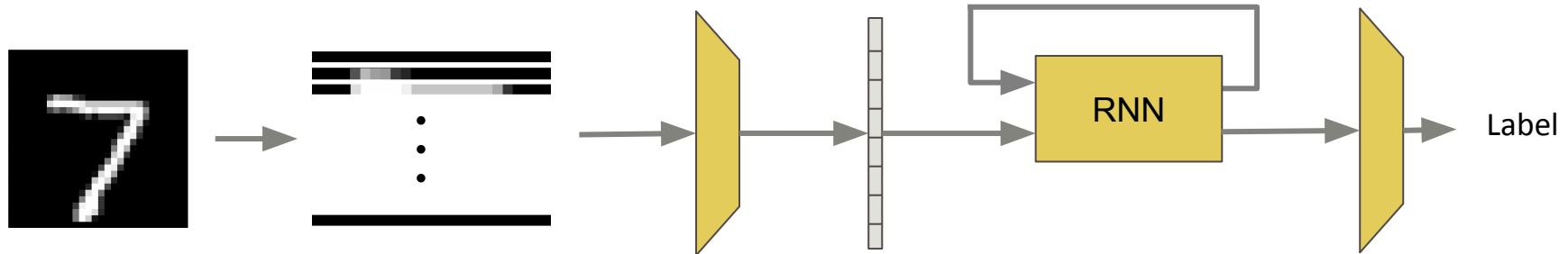
eling is building complex and expressive models that are also tractable and scalable. This trade-off has resulted in a large variety of generative models, each having their advantages. Most work focuses on stochastic latent variable models such as VAE's (Rezende et al., 2014; Kingma & Welling, 2013) that aim to extract meaningful representations, but often come with an intractable inference step that can hinder their performance.

One effective approach to *tractably* model a joint distribu-



MNIST Sequential Classifier

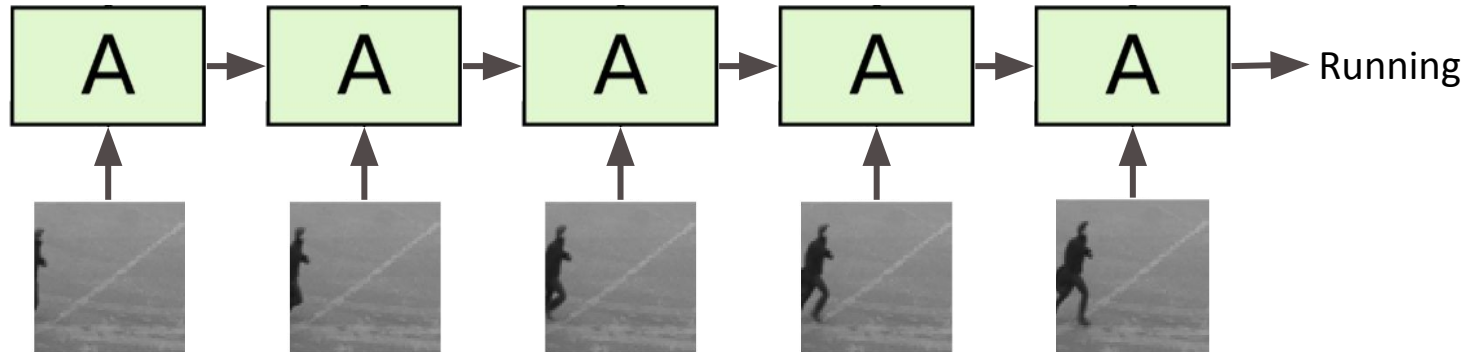
- How to classify images with an RNN?
 1. Split images row-wise into a sequence of 28 28-dim vectors
 2. Embed 28-dim input into vector representation
 3. Sequentially feed embeddings to RNN
 4. Classify RNN output with a fully-connected layer



Sequential Processing of Videos

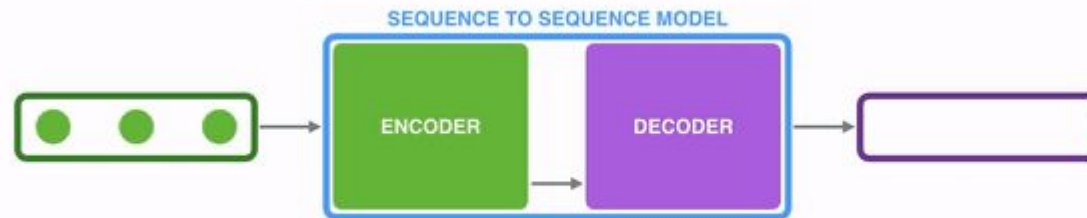
Many-to-One

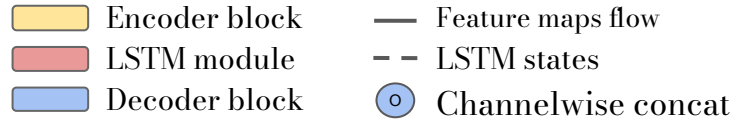
- Maps a sequence of inputs into a single value
 - RNN sequentially processes all inputs
 - Last RNN-output is used to compute the model output
- Applications: action recognition, sentiment classification, ...



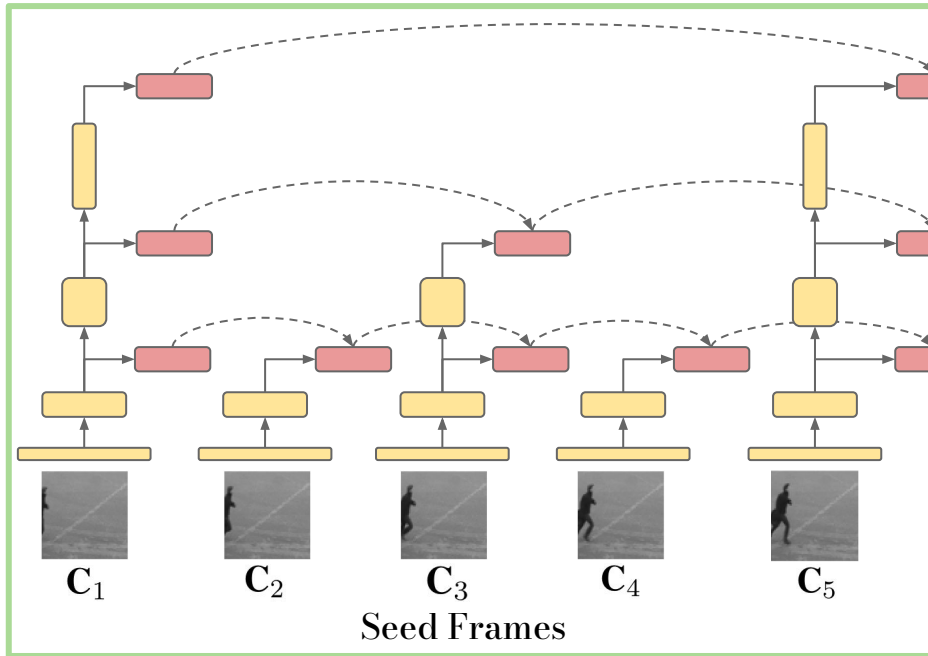
Many-to-Many

- A sequence of inputs is mapped to another sequence:
 - Feed input frames to encoder model
 - Encoded outputs are fed to decoder
 - Decoder obtains final outputs
- Applications: video prediction, action segmentation, motion forecasting, ...

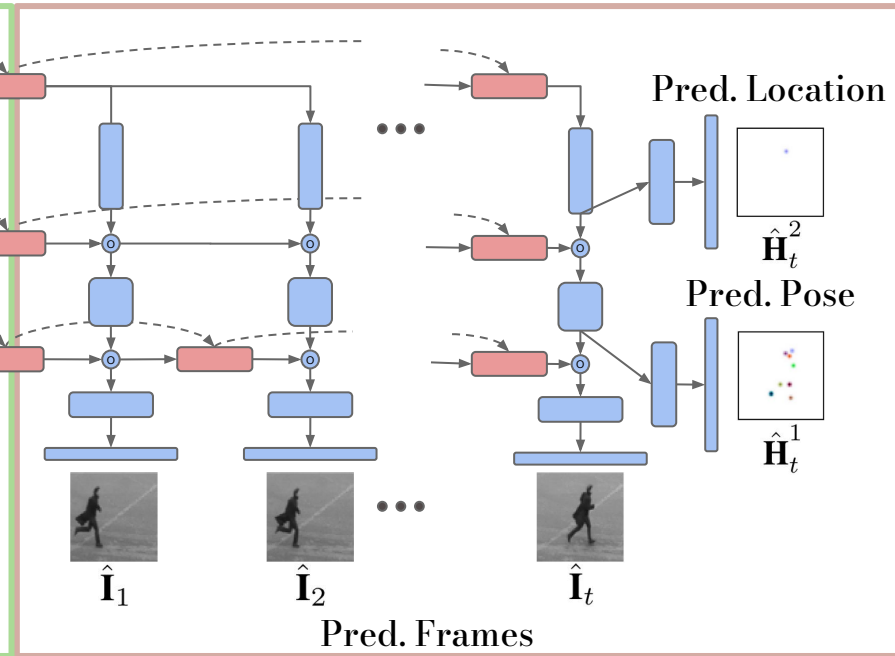




Seed Stage



Prediction Stage



References

1. <https://towardsdatascience.com/all-you-want-to-know-about-deep-learning-8d68dcffc258>
2. Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 9.8 (1997): 1735-1780.
3. Goodfellow, Ian, et al. Deep learning. Vol. 1. No. 2. Cambridge: MIT press, 2016.
4. <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>
5. <https://danijar.com/tips-for-training-recurrent-neural-networks/>
6. Bengio, Yoshua, Patrice Simard, and Paolo Frasconi. "Learning long-term dependencies with gradient descent is difficult." IEEE transactions on neural networks 5.2 (1994): 157-166.
7. Elman, Jeffrey L. "Finding structure in time." Cognitive science 14.2 (1990): 179-211.
8. Gregor, Karol, et al. "Draw: A recurrent neural network for image generation." International Conference on Machine Learning. PMLR, 2015.
9. Ba, Jimmy, Volodymyr Mnih, and Koray Kavukcuoglu. "Multiple object recognition with visual attention." arXiv preprint arXiv:1412.7755 (2014).