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

Recurrent Neural Networks

20.11.2024

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

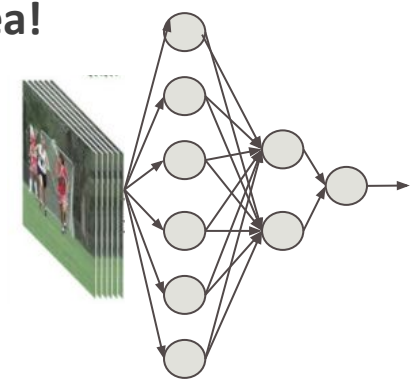
- ‘Snapshots’ are often not informative

  ⇨ 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

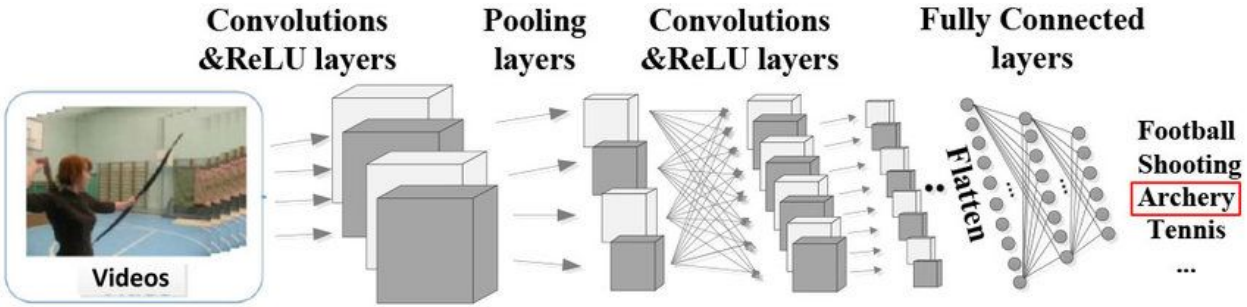


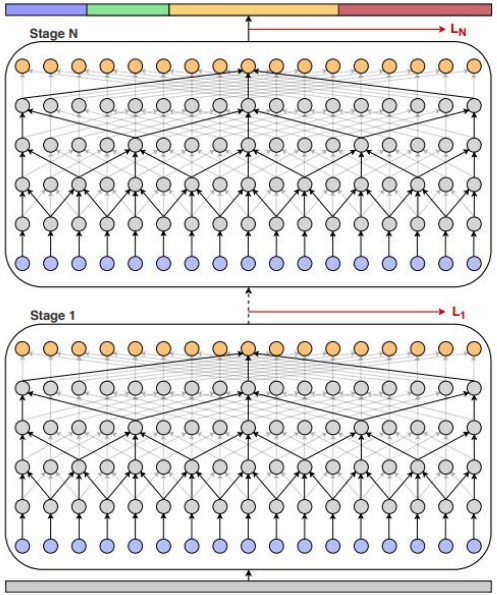
2. Sliding window approaches
   - Process a subset of the inputs at the time

3. Model sequential behavior within the architecture
   - **Recurrent Neural Networks (RNNs)**

Disclaimer I

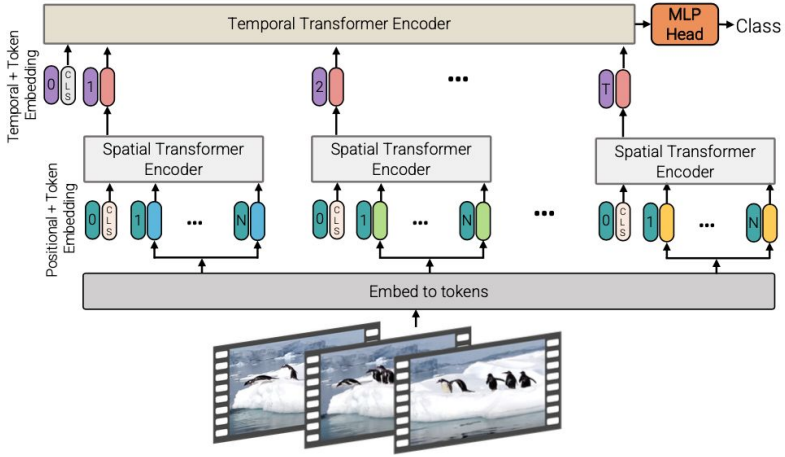
- Powerful CNN-based architectures for video processing
  - 3D-CNNs
  - Temporal Convolutional Networks (TCNs)
- Better than RNNs on many video tasks

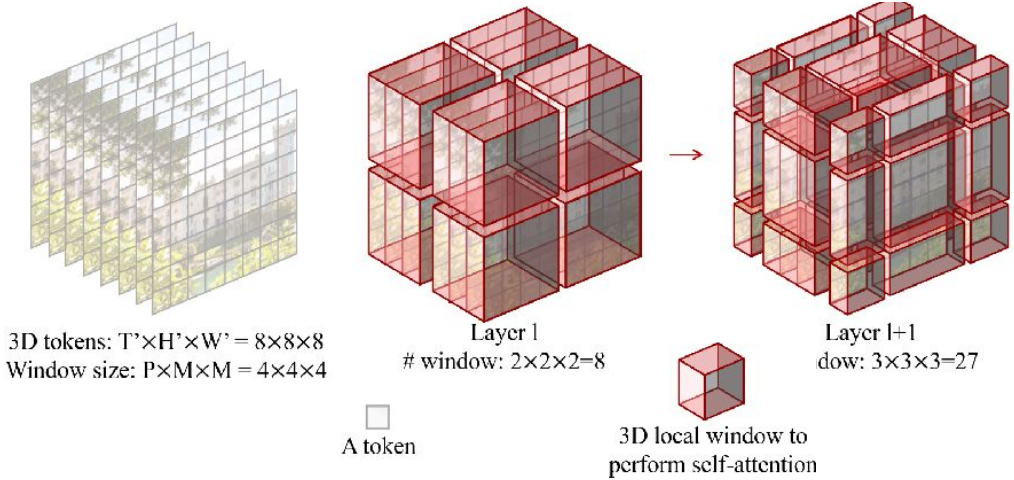




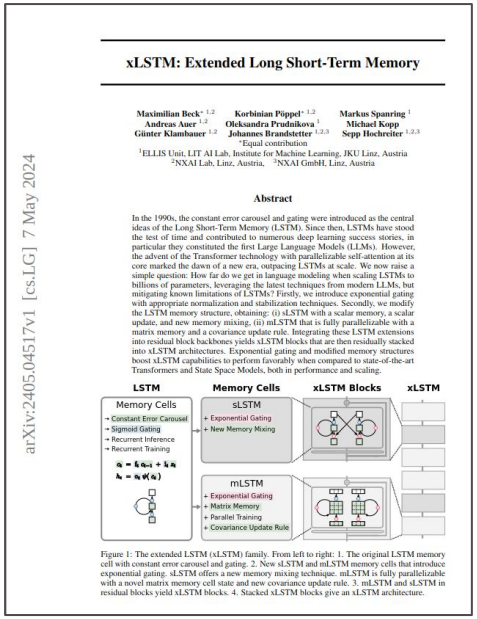
Disclaimer II

- Transformers are powerful architectures for video processing
  - ViViT
  - Video Swin Transformer
- Better than RNNs and 3D-CNNs on many sequence and video tasks





RNNs Still Alive!



arXiv:2405.04517v1 [cs.LG] 7 May 2024

xLSTM: Extended Long Short-Term Memory

Maximilian Beck[*1,2] Korbinian Pöppel[*1,2] Markus Spanring[1]
Andreas Auer[1,2] Oleksandra Prudnikova[1] Michael Kopp
Günter Klambauer[1,2] Johannes Brandstetter[1,2,3] Sepp Hochreiter[1,2,3]
[*]Equal contribution
[1]ELLIS Unit, LIT AI Lab, Institute for Machine Learning, JKU Linz, Austria
[2]NXAI Lab, Linz, Austria, [3]NXAI GmbH, Linz, Austria

Abstract

In the 1990s, the constant error carousel and gating were introduced as the central ideas of the Long Short-Term Memory (LSTM). Since then, LSTMs have stood the test of time and contributed to numerous deep learning success stories, in particular they constituted the first Large Language Models (LLMs). However, the advent of the Transformer technology with parallelizable self-attention at its core marked the dawn of a new era, outpacing LSTMs at scale. We now raise a simple question: How far do we get in language modeling when scaling LSTMs to billions of parameters, leveraging the latest techniques from modern LLMs, but mitigating known limitations of LSTMs? Firstly, we introduce exponential gating with appropriate normalization and stabilization techniques. Secondly, we modify the LSTM memory structure, obtaining: (i) sLSTM with a scalar memory, a scalar update, and new memory mixing, (ii) mLSTM that is fully parallelizable with a matrix memory and a covariance update rule. Integrating these LSTM extensions into residual block backbones yields xLSTM blocks that are then residually stacked into xLSTM architectures. Exponential gating and modified memory structures boost xLSTM capabilities to perform favorably when compared to state-of-the-art Transformers and State Space Models, both in performance and scaling.

Figure 1: The extended LSTM (xLSTM) family. From left to right: 1. The original LSTM memory cell with constant error carousel and gating. 2. New sLSTM and mLSTM memory cells that introduce exponential gating. sLSTM offers a new memory mixing technique. mLSTM is fully parallelizable with a novel matrix memory cell state and new covariance update rule. 3. mLSTM and sLSTM in residual blocks yield xLSTM blocks. 4. Stacked xLSTM blocks give an xLSTM architecture.

arXiv:2406.04303v2 [cs.CV] 2 Jul 2024

Vision-LSTM: xLSTM as Generic Vision Backbone

Benedikt Alkin[1,2] Maximilian Beck[1,2] Korbinian Pöppel[1,2]
Sepp Hochreiter[1,2,3] Johannes Brandstetter[1,2,3]
[1] ELLIS Unit Linz, Institute for Machine Learning, JKU Linz, Austria
[2] NXAI Lab, Linz, Austria
[3] NXAI GmbH, Linz, Austria
{alkin,brandstetter}@ml.jku.at

Abstract

Transformers are widely used as generic backbones in computer vision, despite initially introduced for natural language processing. Recently, the Long Short-Term Memory (LSTM) has been extended to a scalable and performant architecture - the xLSTM - which overcomes long-standing LSTM limitations via exponential gating and parallelizable matrix memory structure. In this report, we introduce Vision-LSTM (ViL), an adaption of the xLSTM building blocks to computer vision. ViL comprises a stack of xLSTM blocks where odd blocks process the sequence of patch tokens from top to bottom while even blocks go from bottom to top. Experiments show that ViL holds promise to be further deployed as new generic backbone for computer vision architectures.
Project page: https://nx-ai.github.io/vision-lstm/

1 Introduction

Language modeling architectures — such as Transformers [56, 1, 50] or more recently State Space Models [24, 25] such as Mamba [23] — are commonly adapted to the domain of computer vision to make use of their powerful modeling capabilities. However, in natural language processing, an input sentence is typically encoded into tokens that represent words or common subwords [8] via a discrete vocabulary. To encode images into a set of tokens, Vision Transformer [18] (ViT) proposed to group an input image into non-overlapping patches (of e.g. 16x16 pixel), linearly project them into a sequence of so-called patch tokens and add positional information to these tokens. This sequence can then be processed by language modeling architectures.

The Extended Long Short-Term Memory (xLSTM) family [5] was recently introduced as a new architecture for language modeling. It demonstrates the resurgence of LSTM in the LLM era, performing favorably against the likes of Transformers and State Space Models (SSMs). Analogous to existing vision versions of Transformers or State Space Models, e.g., ViT [18] or Vision Mamba [67], which have produced great results in various computer vision tasks [48, 37, 44, 46, 3], we introduce Vision LSTM (ViL) – a generic computer vision backbone that uses xLSTM blocks as its core components. To adjust xLSTM (an autoregressive model) to computer vision (an often non-autoregressive domain), we employ a stack of alternating mLSTM blocks [5] where odd blocks process patches row-wise from top left to bottom right and even blocks go from bottom right to top left. This simple alternating design allows ViL to efficiently process non-sequential inputs, such as images, without introducing additional computations.

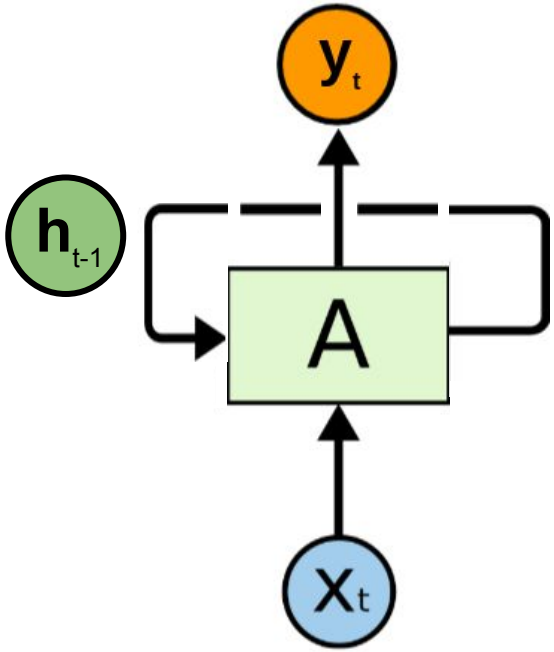
Similar to vision adaptions of SSMs [40, 67, 58], ViL can exhibit linear computational and memory complexity w.r.t. sequence length which makes it appealing for tasks that benefit from high-resolution images such as medical imaging [9, 26, 55, 62], segmentation [37, 10], or physics simulations [6, 43, 7, 2]. In contrast, ViT's computational complexity scales quadratically due to the self-attention mechanism, rendering them costly to apply to high-resolution tasks.

		Single-scale		Multi-scale	
Model	#Params	mIoU	ACC	mIoU	ACC
DeiT-T [51]	10M	38.1	78.2	40.3	79.9
DeiT-III-T [54]	10M	39.8	79.2	42.2	80.7
Vim-T [67]	13M	41.0	-	-	-
ViL-T	11M	**41.2**	**80.2**	**43.1**	**81.3**
DeiT-S [51]	41M	43.1	80.7	45.2	81.8
DeiT-III-S [54]	41M	45.2	81.5	46.3	82.3
Vim-S [67]	46M	44.9	-	-	-
Mamba®-S [58]	56M	45.3	-	-	-
ViL-S	42M	**46.3**	**82.0**	**47.9**	**82.9**
DeiT-B [51]	113M	45.8	82.1	47.0	82.9
DeiT-III-B [54]	113M	47.5	82.6	49.0	83.3
Mamba®-B [58]	132M	47.7	-	-	-
ViL-B	115M	**48.6**	**82.8**	**49.6**	**83.3**

Table 2: Semantic segmentation results on ADE20K [66] using UperNet [61]. We report mean intersection over union (mIoU) and pixelwise accuracy (ACC) for single- and multi-scale evaluation. Models are trained for 160K updates with a batchsize of 16 on 512x512 resolution. Detailed hyperparameters are listed in Appendix Table 14.

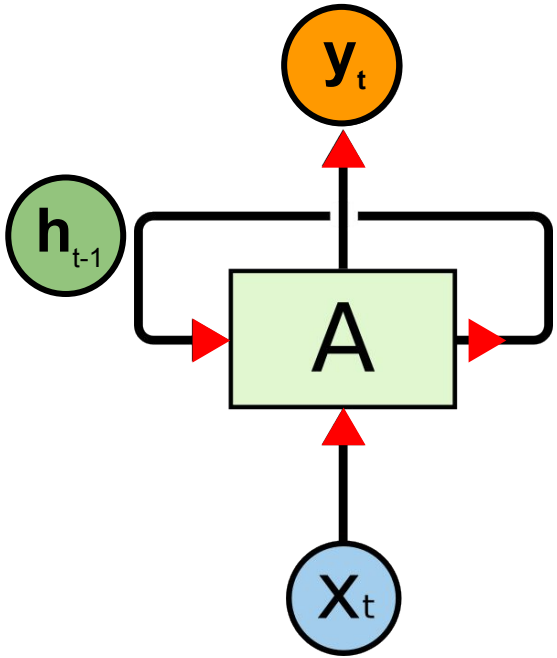
Simple RNNs

Simple RNNs



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

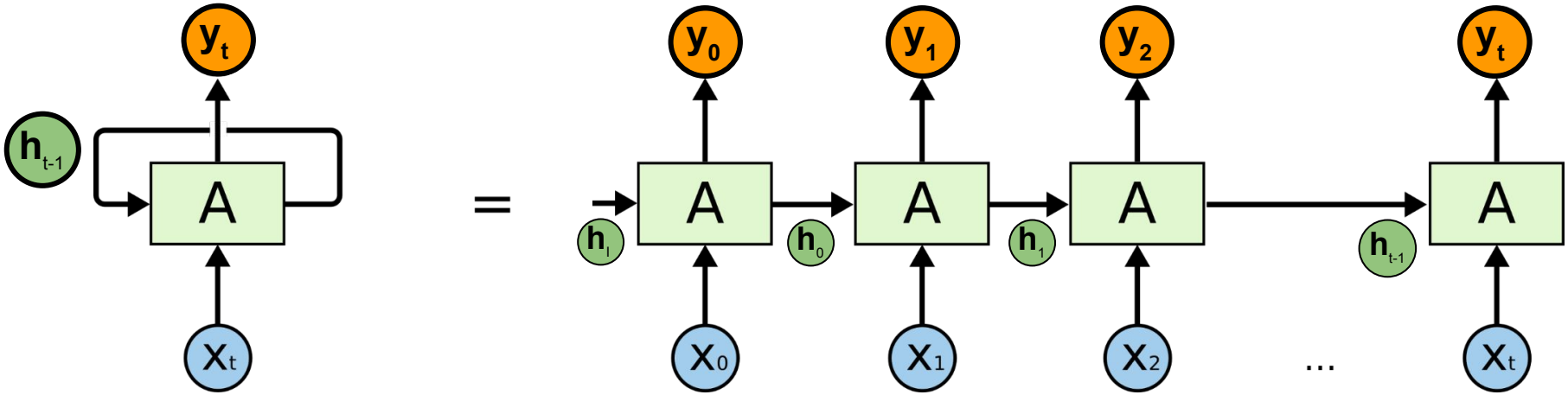
RNN Structure



- **Input** at time *t:* $\mathbf{X}_t$ is feed to a network
- **Extra input: Hidden state** $\mathbf{h}_{t-1}$ obtained by the unit
- **Outputs**
  - $\mathbf{Y}_t$: output with information from present and past
  - $\mathbf{h}_t$: 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



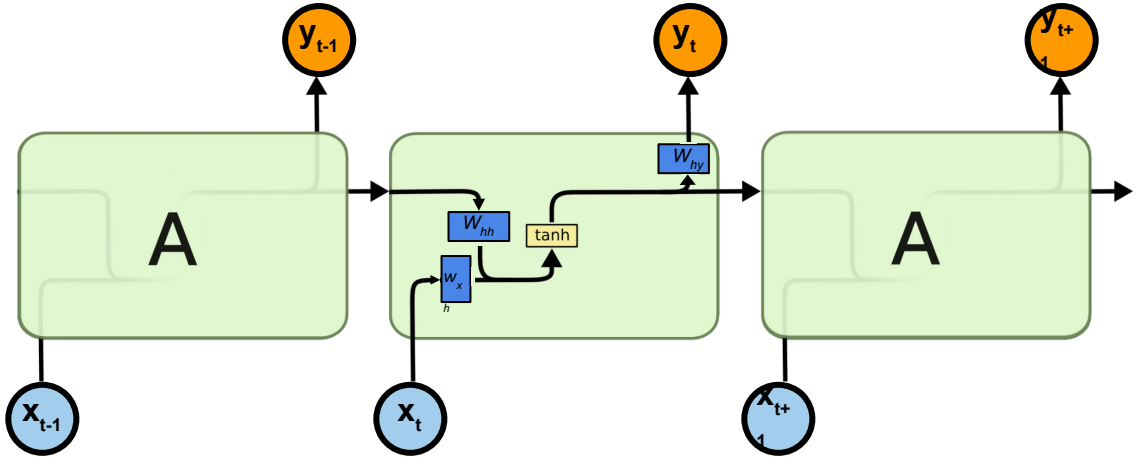
Vanilla RNN

- 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?

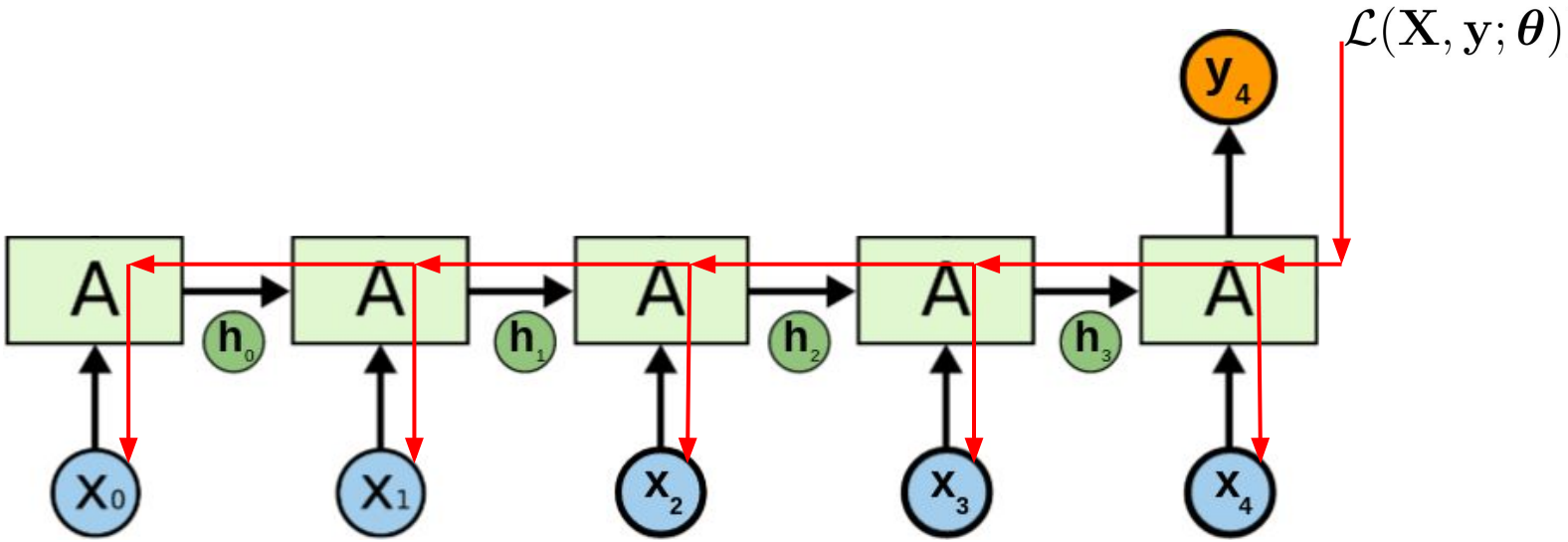
Vanilla RNN

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

```
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

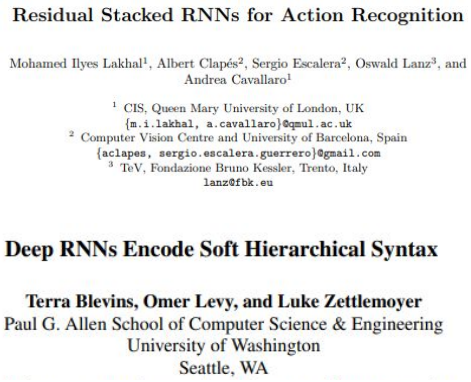
- RNNs are trained with backpropagation through time (BPTT)



Deep RNN

- We can stack several RNNs
  - Similar to CNNs

- Very common to use stack least two RNNs



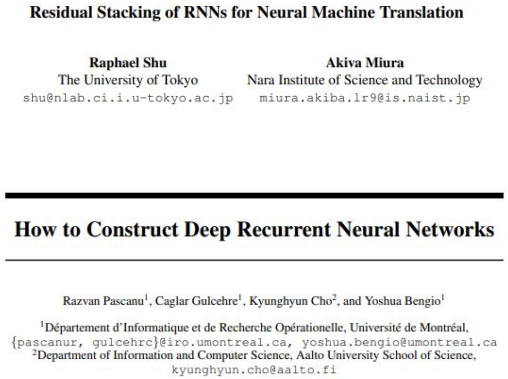
Residual Stacked RNNs for Action Recognition

Mohamed Ilyes Lakhal[1], Albert Clapés[2], Sergio Escalera[2], Oswald Lanz[3], and Andrea Cavallaro[1]

[1] CIS, Queen Mary University of London, UK
{m.i.lakhal, 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

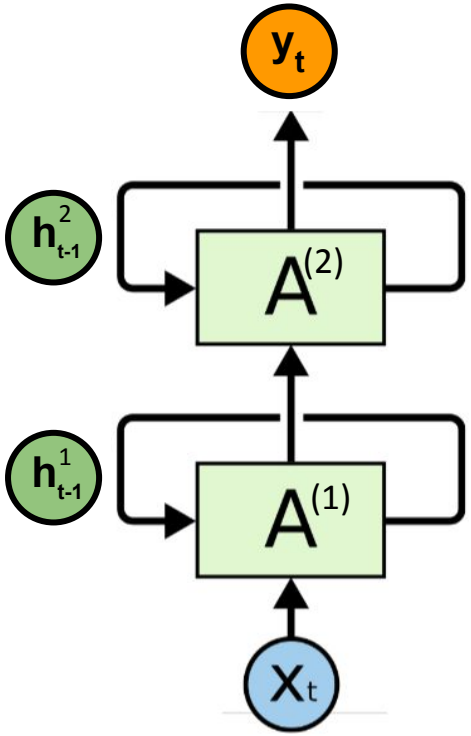
Raphael Shu
The University of Tokyo
shu@nlab.ci.i.u-tokyo.ac.jp

Akiva Miura
Nara Institute of Science and Technology
miura.akiba.lr9@is.naist.jp

How to Construct Deep Recurrent Neural Networks

Razvan Pascanu[1], Caglar Gulcehre[1], Kyunghyun Cho[2], and Yoshua Bengio[1]

[1]Département d'Informatique et de Recherche Opérationelle, Université de Montréal,
{pascanur, gulcehrc}@iro.umontreal.ca, yoshua.bengio@umontreal.ca
[2]Department of Information and Computer Science, Aalto University School of Science,
kyunghyun.cho@aalto.fi



Problem with RNNs

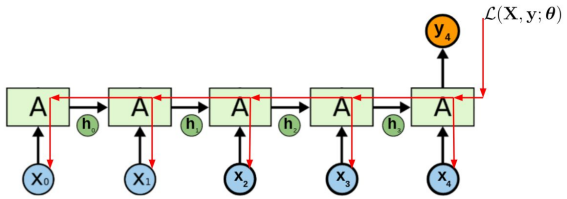
Gradient Problem

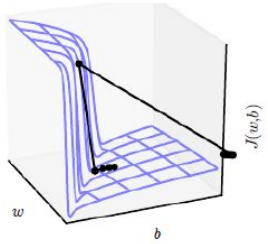
- Weights/Gradients are computed by a deep cascade of matrix multiplications and TanH nonlinearities

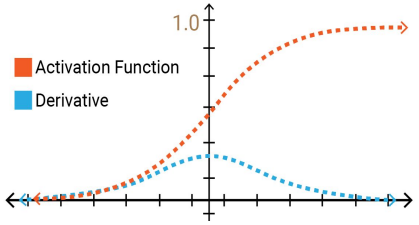
⇒ Gradients prone to vanishing or exploding

- **Exploding gradient:** just clip the gradients

- **Vanishing gradient:** hard to solve







Memory Overwriting

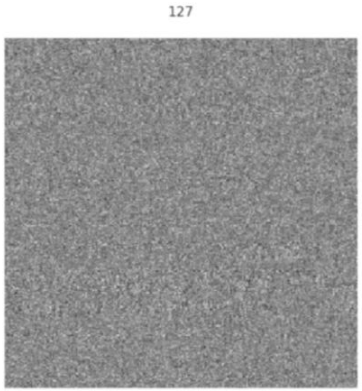
- RNNs have it difficult to connect past and present for long time spans

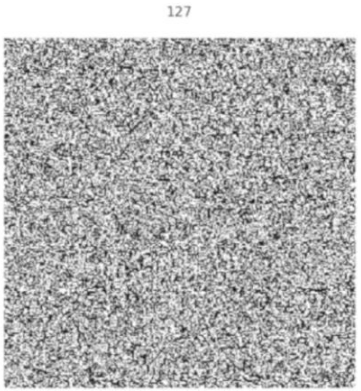
- Hidden state is overwritten every time step

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

Q: Can we do better?

A: Yes!



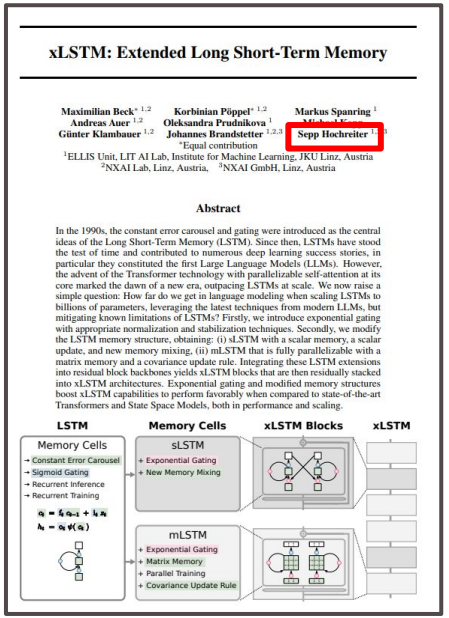


Long Short-Term Memory (LSTM)

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

Newer version from 2024!



xLSTM: Extended Long Short-Term Memory

Maximilian Beck[*,1,2], Korbinian Pöppel[*,1,2], Markus Spanring[1], Andreas Auer[1,2], Oleksandra Prudnikova[1], Johannes Brandstetter[1,2,3], Sepp Hochreiter[1,3], Günter Klambauer[1,2]

[*]Equal contribution

[1]ELLIS Unit, LIT AI Lab, Institute for Machine Learning, JKU Linz, Austria

[2]NXAI Lab, Linz, Austria, [3]NXAI GmbH, Linz, Austria

Abstract

In the 1990s, the constant error carousel and gating were introduced as the central ideas of the Long Short-Term Memory (LSTM). Since then, LSTMs have stood the test of time and contributed to numerous deep learning success stories, in particular they constituted the first Large Language Models (LLMs). However, the advent of the Transformer technology with parallelizable self-attention at its core marked the dawn of a new era, outpacing LSTMs at scale. We now raise a simple question: How far do we get in language modeling when scaling LSTMs to billions of parameters, leveraging the latest techniques from modern LLMs, but mitigating known limitations of LSTMs? Firstly, we introduce exponential gating with appropriate normalization and stabilization techniques. Secondly, we modify the LSTM memory structure, obtaining: (i) sLSTM with a scalar memory, a scalar update, and new memory mixing, (ii) mLSTM that is fully parallelizable with a matrix memory and a covariance update rule. Integrating these LSTM extensions into residual block backbones yields xLSTM blocks that are then residually stacked into xLSTM architectures. Exponential gating and modified memory structures boost xLSTM capabilities to perform favorably when compared to state-of-the-art Transformers and State Space Models, both in performance and scaling.

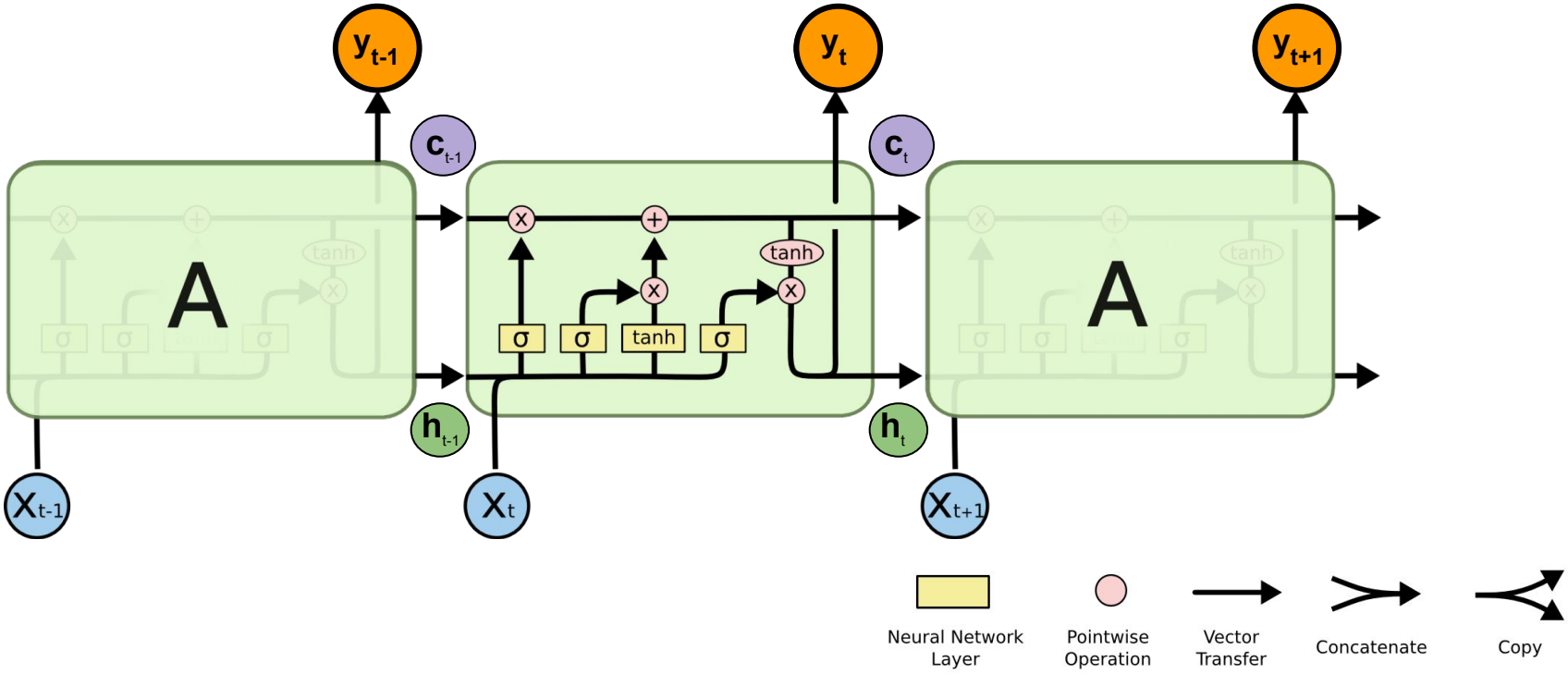
LSTM | Memory Cells | xLSTM Blocks | xLSTM

Memory Cells
→ Constant Error Carousel
→ Sigmoid Gating
→ Recurrent Inference
→ Recurrent Training

sLSTM
+ Exponential Gating
+ New Memory Mixing

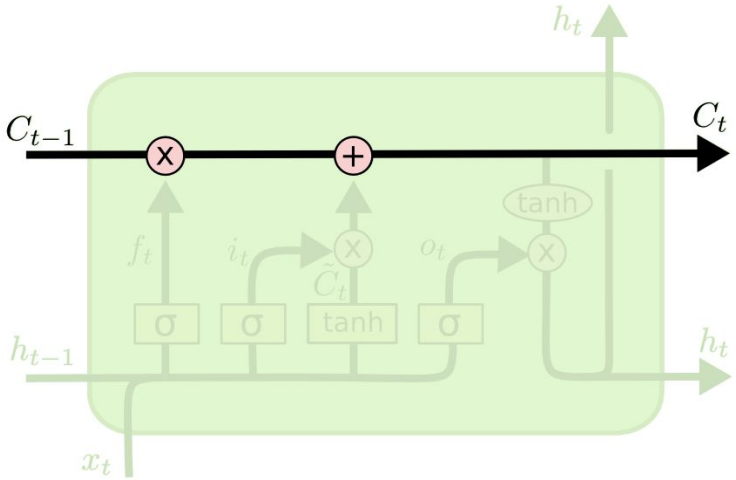
mLSTM
+ Exponential Gating
+ Matrix Memory
+ Parallel Training
+ Covariance Update Rule

LSTM Structure



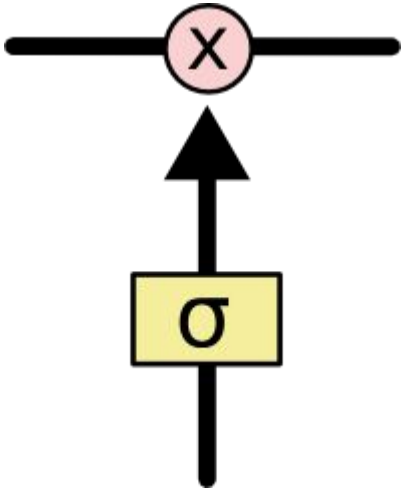
Cell State

- Runs across the entire LSTM with just some minor linear interaction
- LSTM modifies cell state using the so-called gates
- Cell state (*C*) ≠ 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

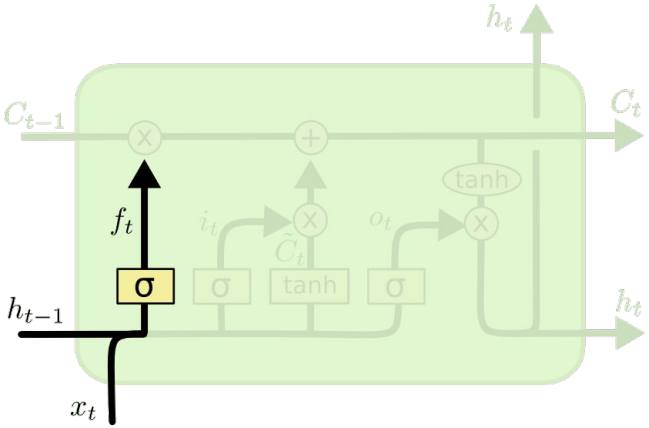


Step-by-Step LSTM Walk Through

Forget Gate

- Decides which information is removed from the cell state

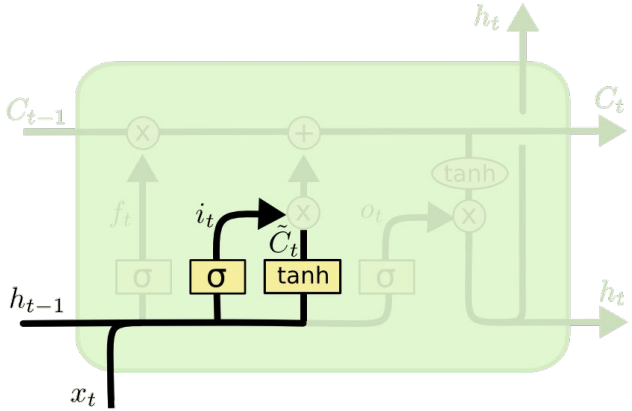
- Input and hidden state determine which values in cell state are forgotten



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

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



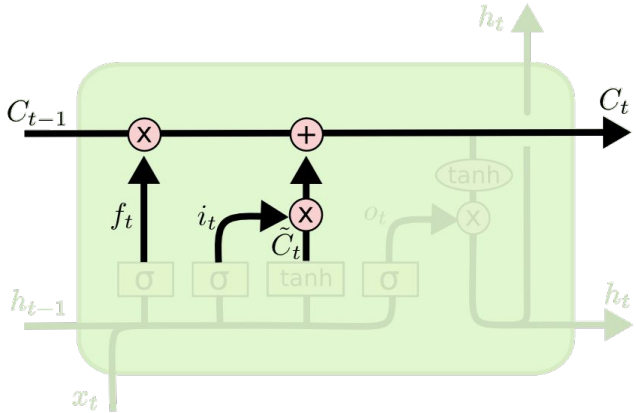
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

Update Gate

- Updates the old cell state into the new cell state

- Sequentially applies the results from the *forget* and *input* gates

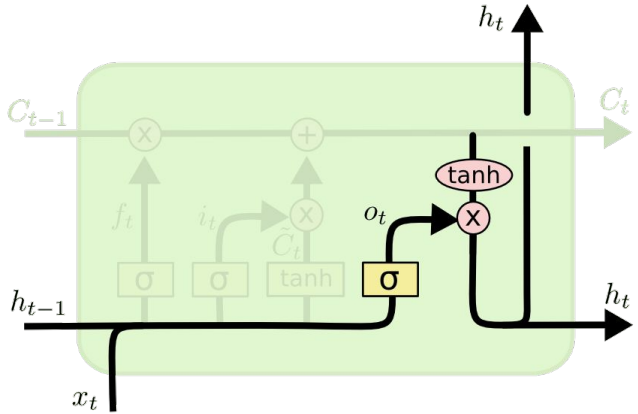


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

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



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

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

Summary

- Forget Gate

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

- Input Gate

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

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

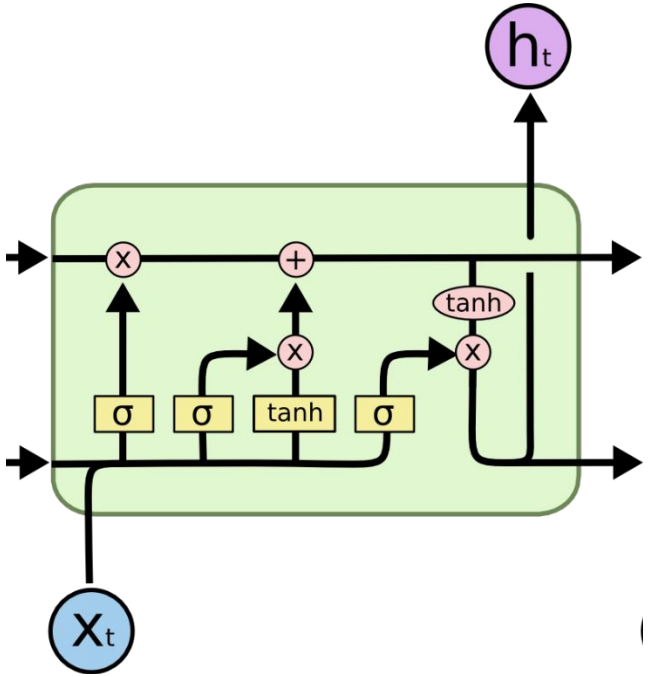
- Update Gate

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

- Output Gate

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

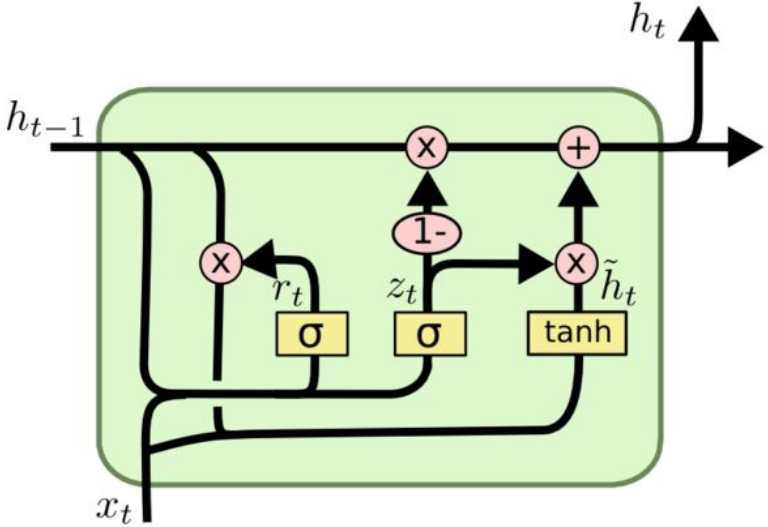
$$h_t = o_t * \tanh\left(C_t\right)$$



LSTM Variants

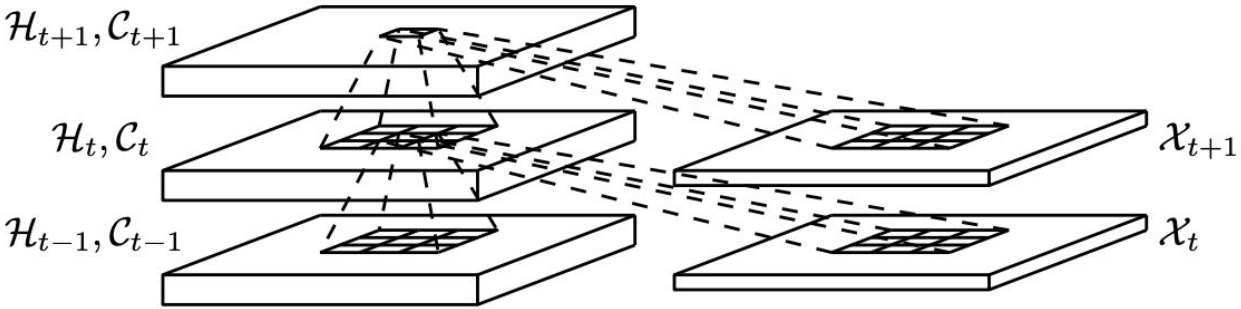
Gated Recurrent Unit

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



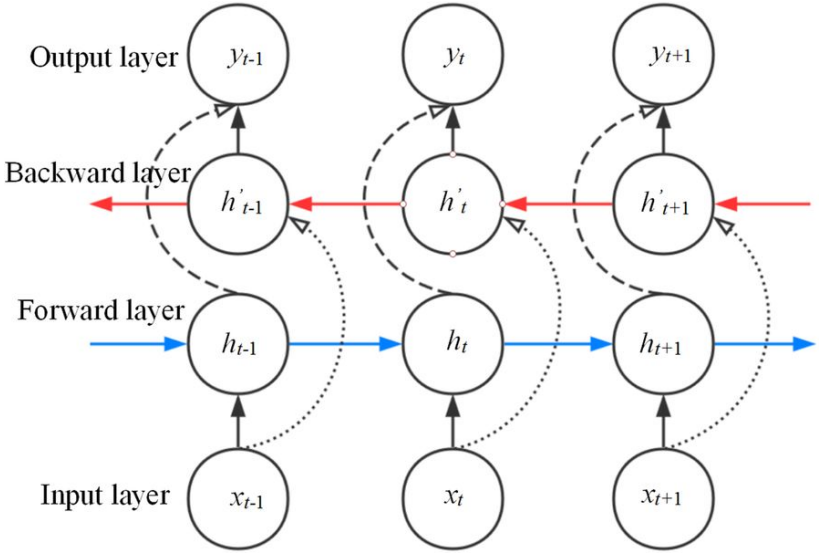
Conv-LSTM

- 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

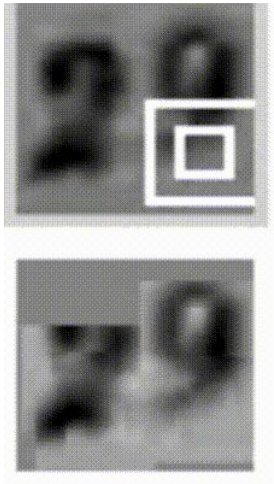
Jimmy Lei Ba[*]
University of Toronto
jimmy@psi.utoronto.ca

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

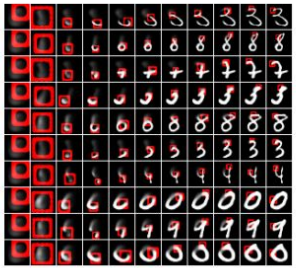
- RNNs can also be applied to datasets with fixed inputs, such as images

DRAW: A Recurrent Neural Network For Image Generation

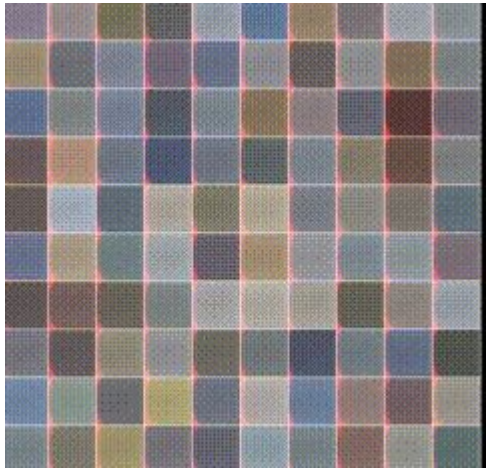
Karol Gregor KAROLG@GOOGLE.COM
Ivo Danihelka DANIHELKA@GOOGLE.COM
Alex Graves GRAVESA@GOOGLE.COM
Danilo Jimenez Rezende DANILOR@GOOGLE.COM
Daan Wierstra WIERSTRA@GOOGLE.COM
Google DeepMind

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



RNNs for Images

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

Pixel Recurrent Neural Networks

Aäron van den Oord AVDNOORD@GOOGLE.COM
Nal Kalchbrenner NALK@GOOGLE.COM
Koray Kavukcuoglu KORAYK@GOOGLE.COM

Google DeepMind

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.

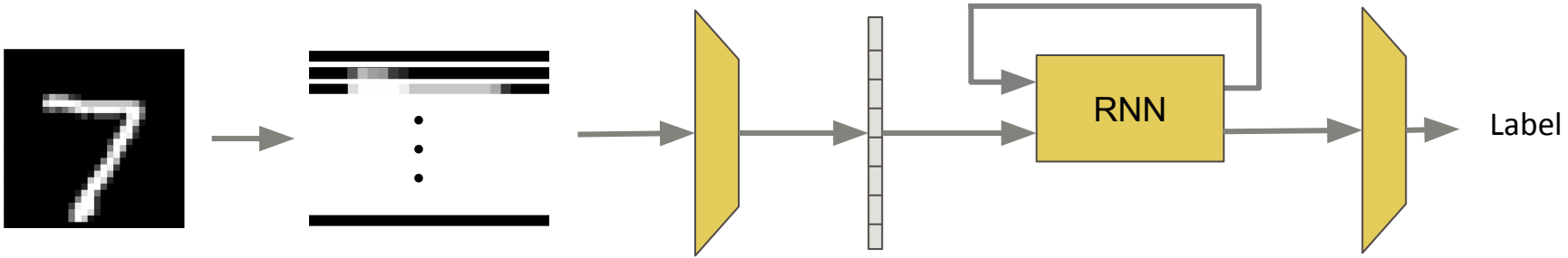
Figure 1. Image completions sampled from a PixelRNN.

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

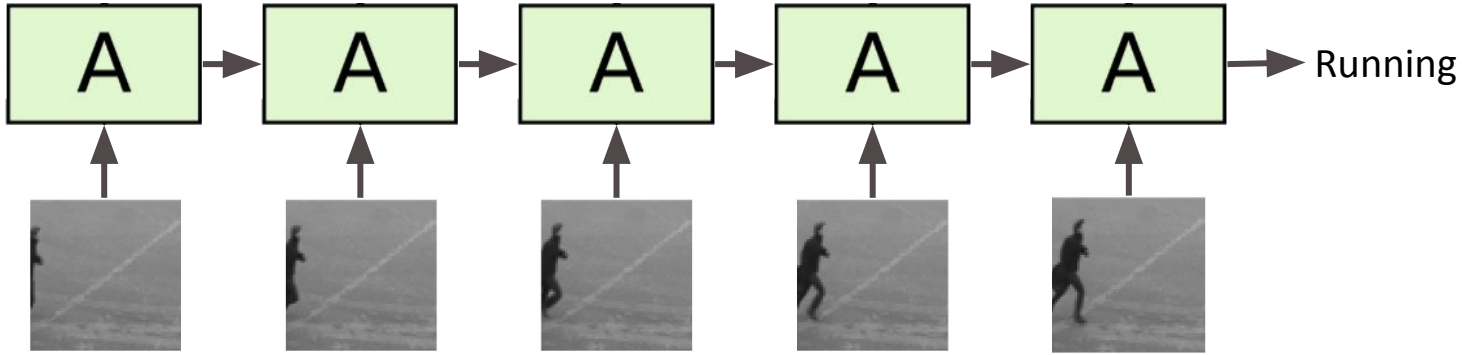


Sequential Processing of Videos

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

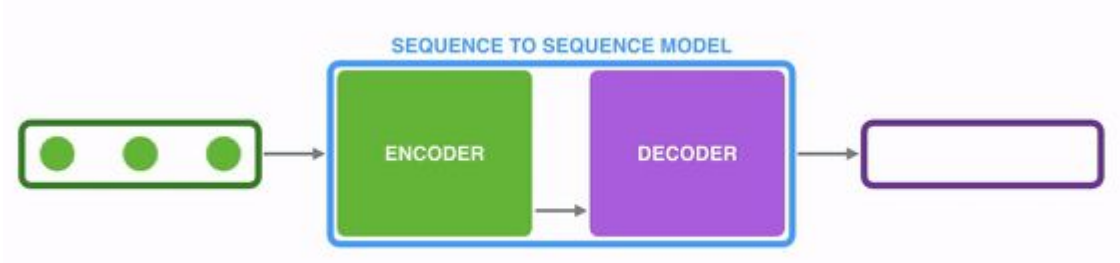
- Applications: action recognition, sentiment classification, …

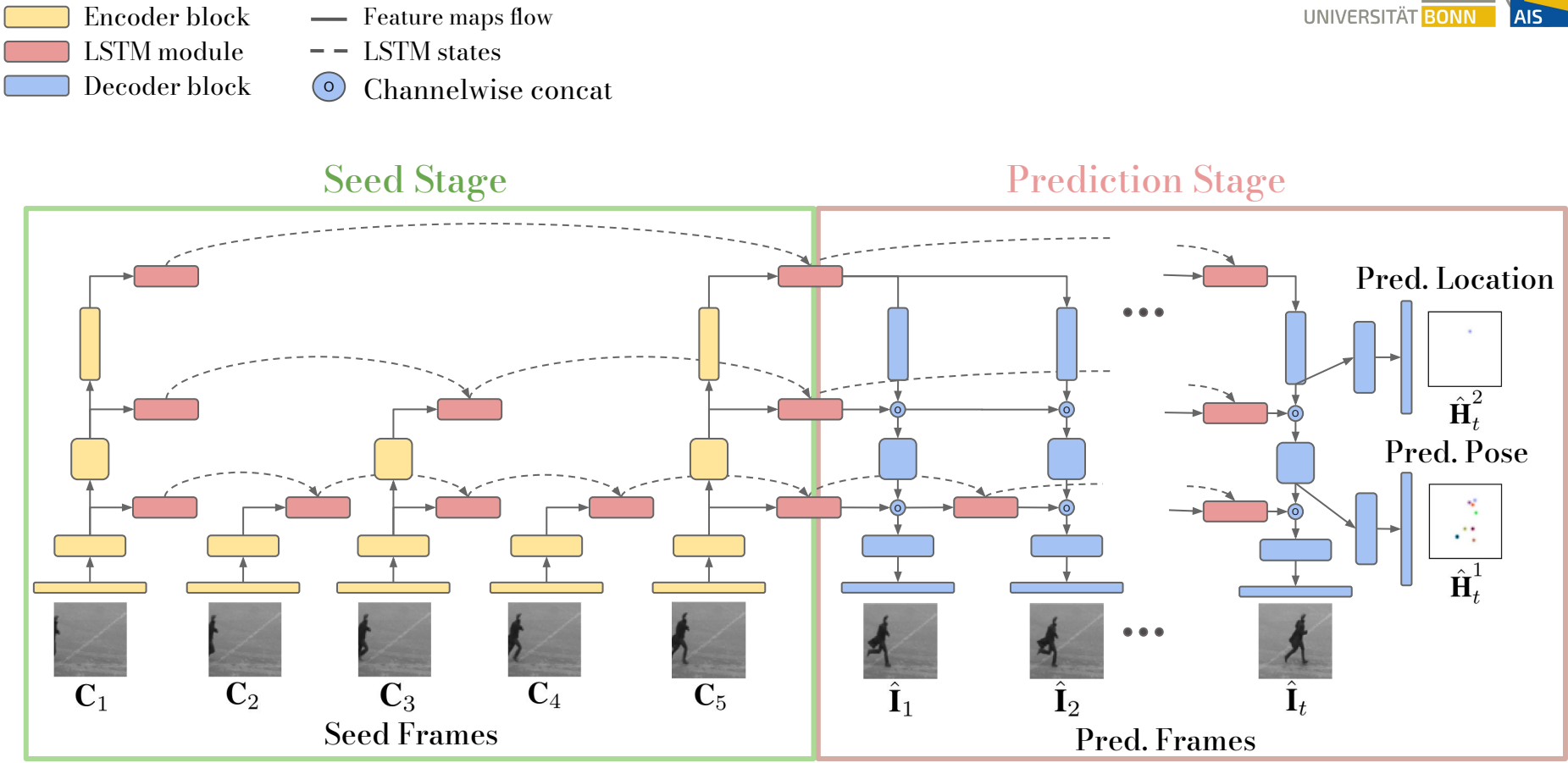


Many-to-Many

- 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, …




Encoder block
LSTM module
Decoder block
Feature maps flow
LSTM states
Channelwise concat
Seed Stage
Prediction Stage
Pred. Location
Pred. Pose
$\hat{\mathbf{H}}_t^2$
$\hat{\mathbf{H}}_t^1$
$\mathbf{C}_1$
$\mathbf{C}_2$
$\mathbf{C}_3$
$\mathbf{C}_4$
$\mathbf{C}_5$
Seed Frames
$\hat{\mathbf{I}}_1$
$\hat{\mathbf{I}}_2$
$\hat{\mathbf{I}}_t$
Pred. Frames

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).