Tutorials on Recurrent Neural Networks and Signature Verification

Muhammad Imran Malik

Marcus Liwicki

Introduction

- Thank you for coming!

- You will hear more about:
  - Neural networks, RNNs
  - Recent architectures (LSTM)
  - Signature verification
  - Forensic Handwriting Examiners (FHEs) work
  - Computational Approaches
  - Comparison between Human and FHEs

- Everything available online:
  - http://www.dfki.uni-kl.de/~liwicki/2013-Szeged.zip

Schedule

Recurrent Neural Networks 10:00-13:00 (Marcus)

- **10:00 Introduction**
- **10:15 RNN and LSTM**
  - Motivation
  - History
  - Recurrent Neural Network training
  - The most powerful RNN: Long Short-Term Memory Networks (LSTM)
- **12:00 Do-It-Yourself**
  - Toolkits, how to apply them, how to use them for your research.

Friday afternoon session – Forensic Handwriting Examination Perspective 14:30-18:00

- **14:30 FHE in general (Marcus)**
  - How forensic experts make comparisons (similarities versus differences, subjectivity)
  - Natural variation, Line quality, Quality versus quantity etc.
  - What forensic experts need from the document analysis community
  - What the document analysis community needs to understand about our work
  - Existing systems and system problems
  - Conclusion scales, Bayesian framework
  - Strength of evidence

Schedule

- **15:30 Tools for FHE**
  - Existing forensic systems: FISH, WANDA, CEDAR-FOX, FBI-system
  - Searching a database of threatening letters
- **16:00 Signatures: Simulation & Disguise Hands-on session (Imran)**
  - Defining signature verification (FHEs-perspective)
  - Problems with signatures (in depth)
  - Showing some example cases
  - Proficiency tests, problems for FDEs
  - Expert results on La Trobe test 2002 & 2006
- **Hands on Session – real Case Work**

Saturday morning session – Automated Signature Verification 10:00-13:00

- **10:00 History of automatic SV (Imran)**
  - Defining signature verification (PR-perspective)
  - Modes of performing verification (online vs. offline)
  - Related work, State-of-the-Art, Evaluation
- **11:00 Current Offline and Online SV Systems (Imran)**
  - Data processing, Features
  - Combined online and offline features
  - Feature subset selection
  - Classification Methods
  - Recent efforts (uniting the perspectives of PR-researchers and FHEs)

Schedule

- **12:00 Comparison between Man and Machine (Imran)**
  - Highlighting machine potential to assist humans
- **12:30 Plenary Discussion (everyone)**
- **13:00 Concluding remarks**

Intro of Marcus Liwicki

- 2001-2004 Berlin – Master of CS
- 2004-2007 Bern – Dr. and PhD (Horst Bunke)
- 2008- Researcher, lecturer at DFKI (Andreas Dengel)
- 2009-2010 JSPS research fellow at Kyushu University (Seiichi Uchida)
- 2011 finished habilitation – current title: Privatdozent
- HWR, Knowledge Management, Forensics, Neural Nets, HCI, IUI
- More than 90 publications including one book, three book chapters and 15 journal papers
- Third-party funds > 2 Mio € (6 Mio € in progress)
  - EU, DFG, BMBF, BMWi, Federal Funding, Industry, …
- Research Group 4+3 PhD, 7 others (>35 supervised in past)

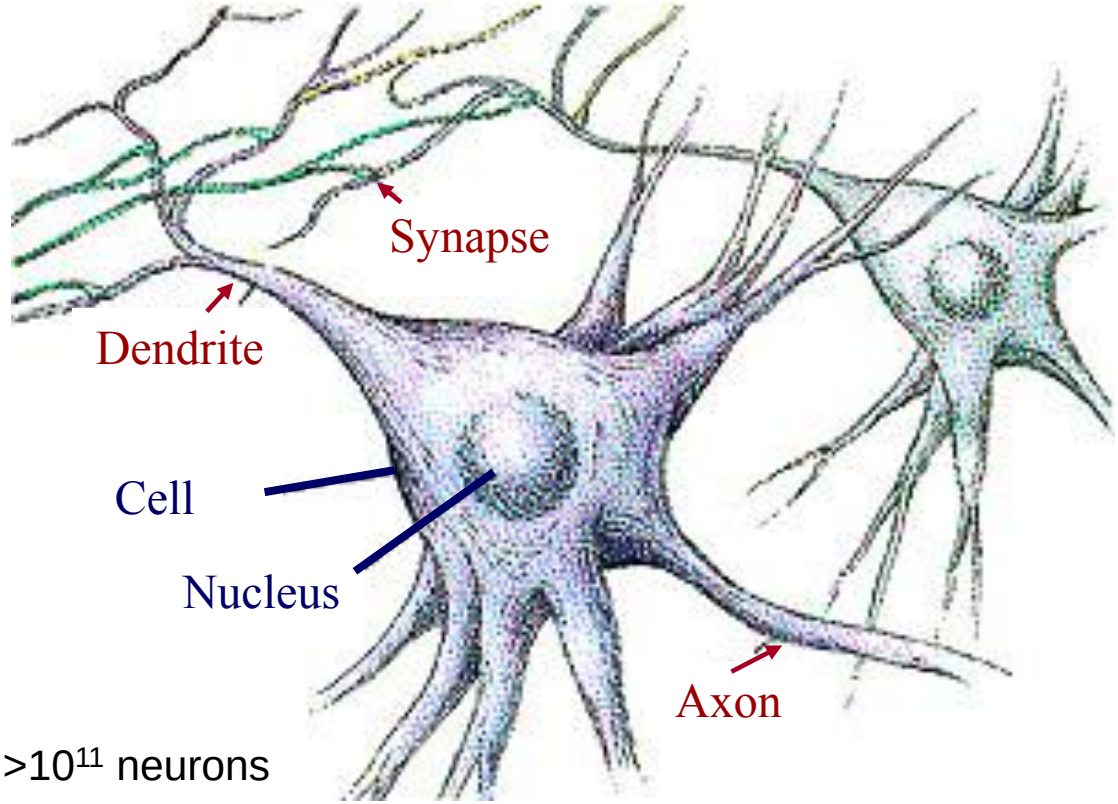
Recurrent and BLSTM Neural Networks

Dr. Marcus Eichenberger-Liwicki
DFKI, Germany
University of Fribourg, Switzerland
Marcus.Liwicki@dfki.de

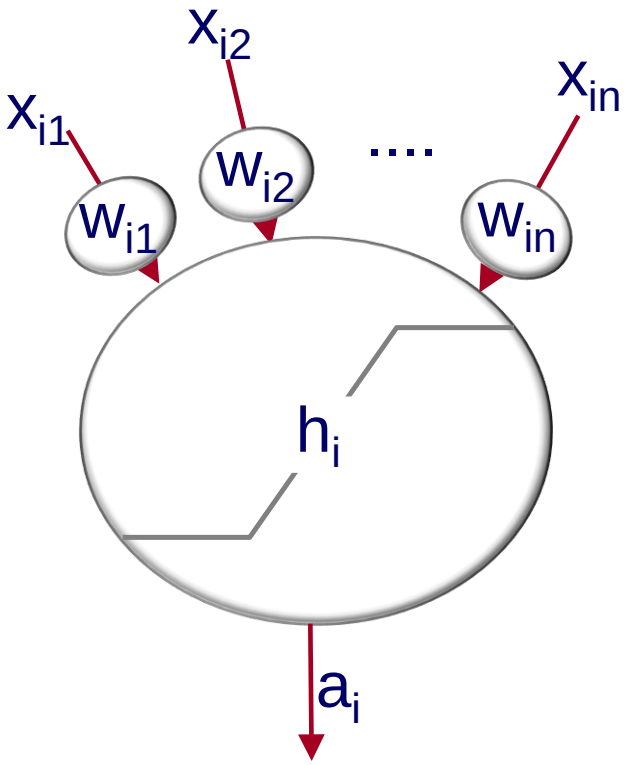
B

Bebop

A Bebop vagy bop a jazz-zene egyik stílusa, ami gyors tempójáról és virtuóz hangszeres improvizációiról ismert.



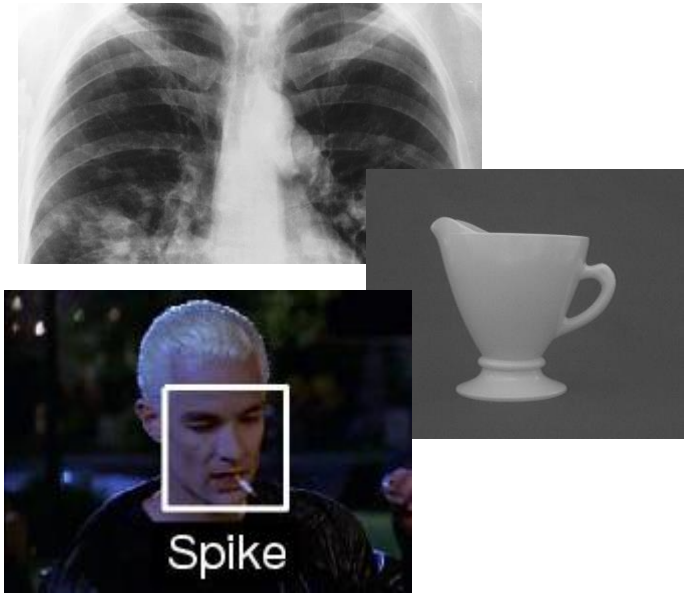
>10^{11} neurons



Goal: make computers intelligent
Idea: simulate neural behavior on PC

Applications: Recognize Patterns

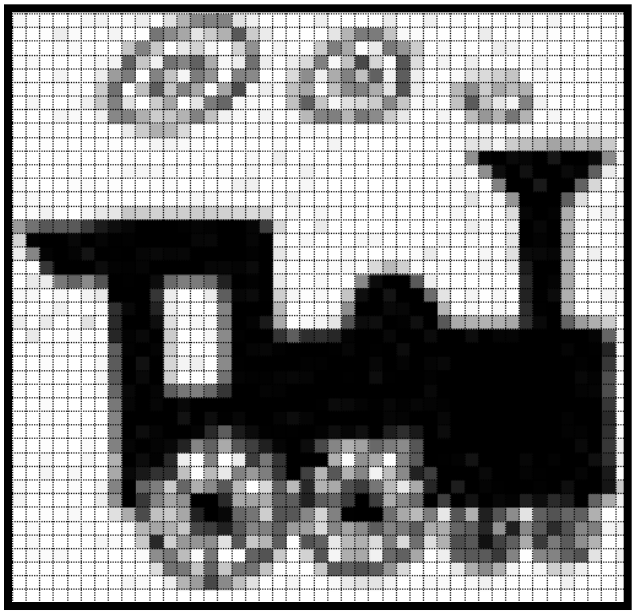
- Image Analysis
  - Detection (e.g., disease)
  - Recognition (e.g., objects)
  - Identification (e.g., persons)
- Data Mining
  - Classification
  - Change and Deviation Detection
  - Knowledge Discovery
- Prognosis
  - Ozone prognosis
  - Weather Forecast
  - Stock market prediction
- Games, …

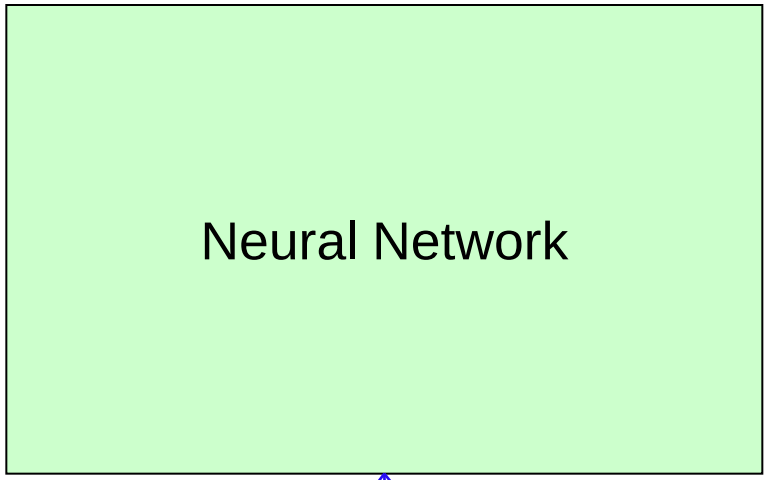


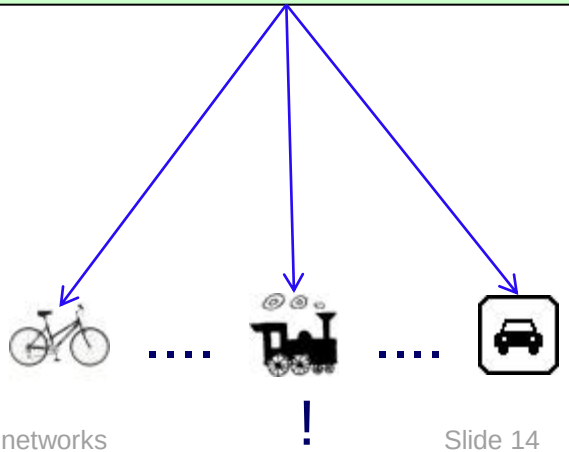
Outline

1. Motivation
2. Multi-Layer Perceptrons (MLP) and their Limits
3. Recurrent Neural Networks (RNN)
4. Long Short-Term Memories (LSTM)
5. Bidirectional LSTM
6. Related Architectures
7. Summary

- Object image

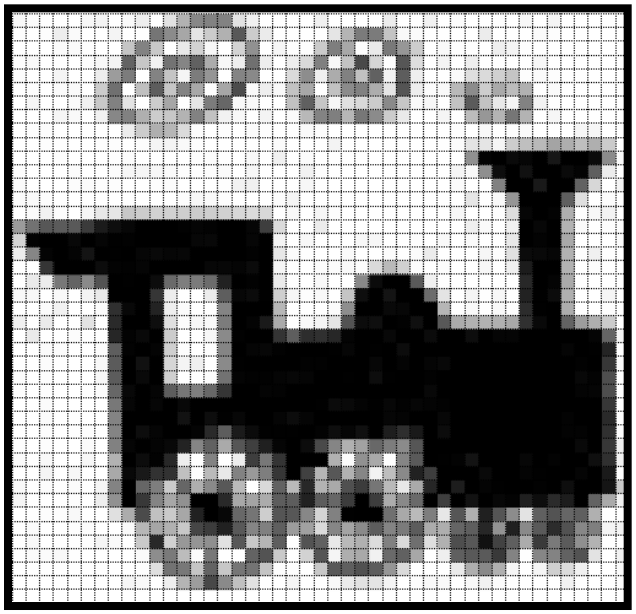






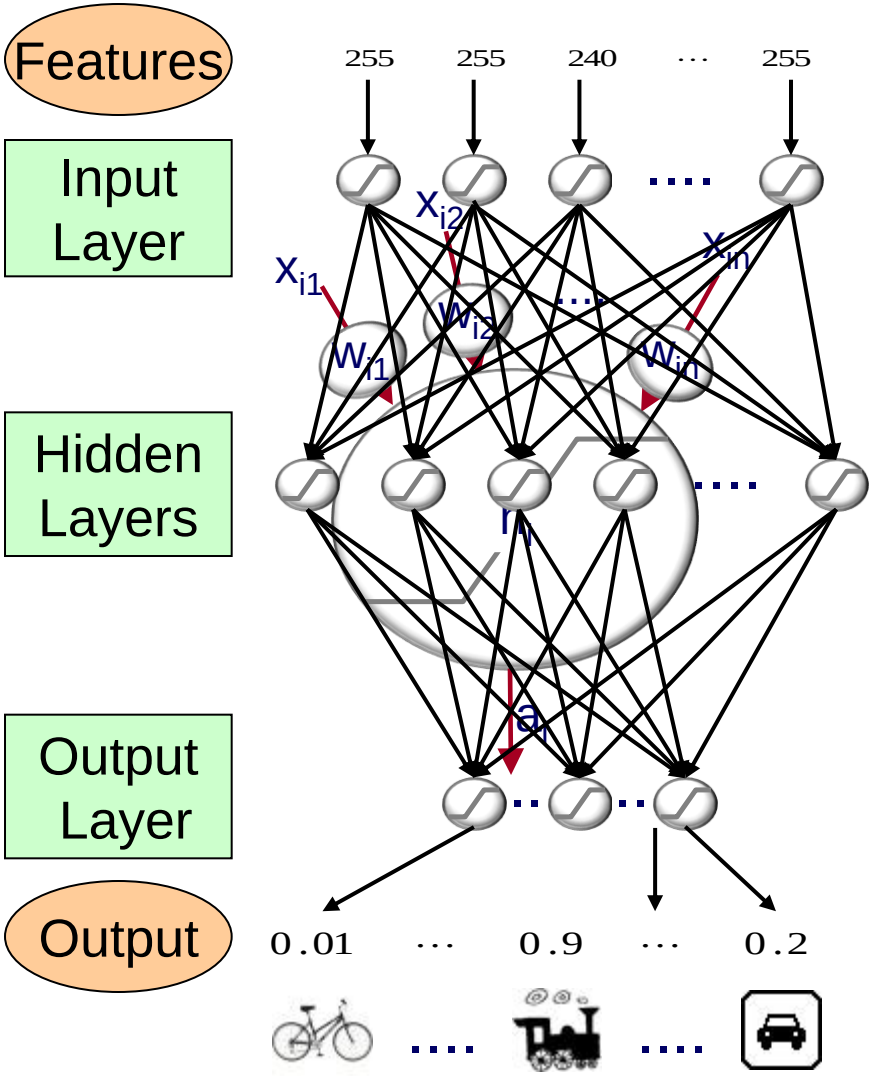
Multi-Layer Perceptrons for Object Recognition

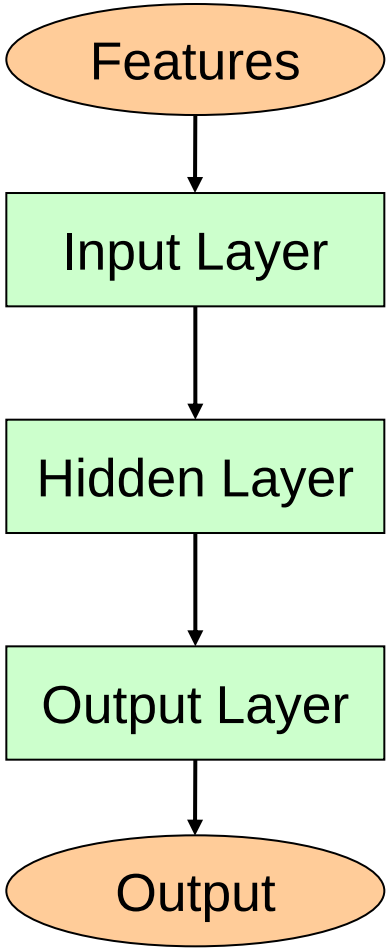
- Object image



- Represent with numbers

255	255	240	⋯	255
255	248	232	⋯	255
252	247	238	⋯	239
⋮	⋮	⋮	⋱	⋮
255	255	255	⋯	255





- Feature vector (at timestamp t)

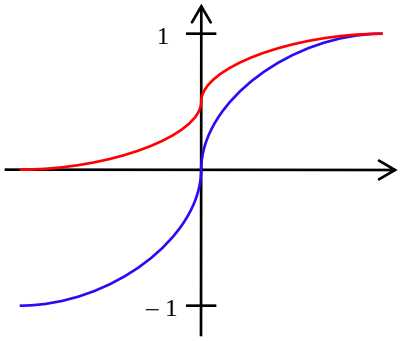
$$x_1^t, \ldots, x_n^t$$

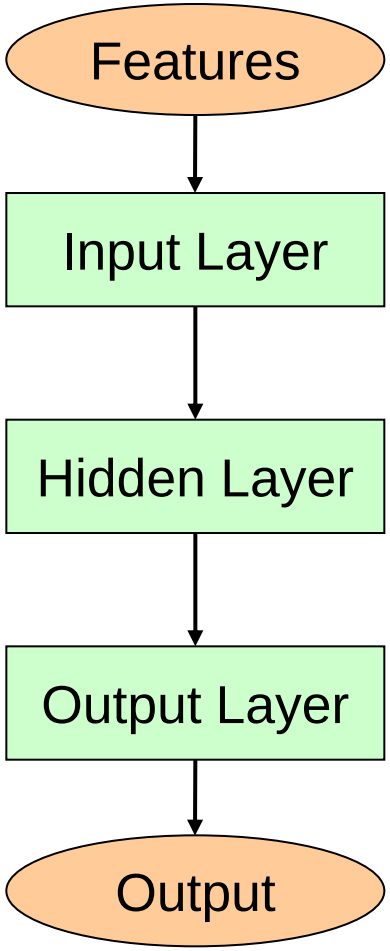
- Perceptrons in the individual layers
  - Aggregation function $a^t = \sum w_i x_i^t$
  - Activation function (squashing f.)

$$b_h^t = h(a^t)$$

$$h: \quad \sigma(x) = \frac{1}{1+e^{-x}}$$

$$\tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$$

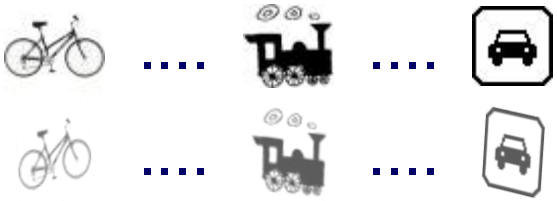




- For every hidden layer:

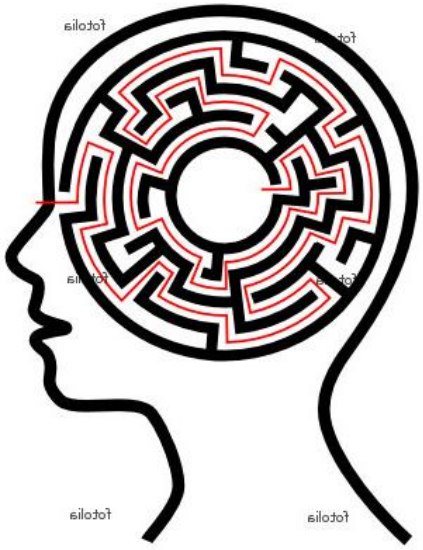
$$b_h^t = h(\sum w_{h'h} b_{h'}^t)$$

- Output depends on the **weights**
- Training of weights by using the backpropagation algorithm
  - Set of training samples with ground truth
  - Apply the network on training samples
  - Propagate error of desired outputs back
  - Update the weights into the opposite direction of the error gradient



- Up to now: static input/output operation

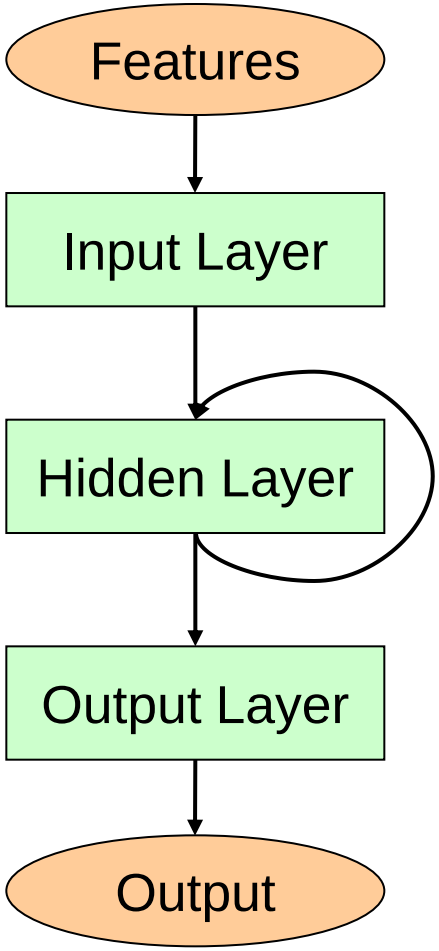
$$x_1, \ldots, x_n \rightarrow y$$



- Human brain is capable of memorizing
- Needed for solving many problems
  - Sequence recognition
  - Navigation through a labyrinth
  - Video analysis

$$\left((x_1^1, \ldots, x_n^1), \ldots, (x_1^T, \ldots, x_n^T)\right) \rightarrow (y^1, \ldots, y^U) \mid U \leq T$$

- Idea: add backward-connections to keep an internal state

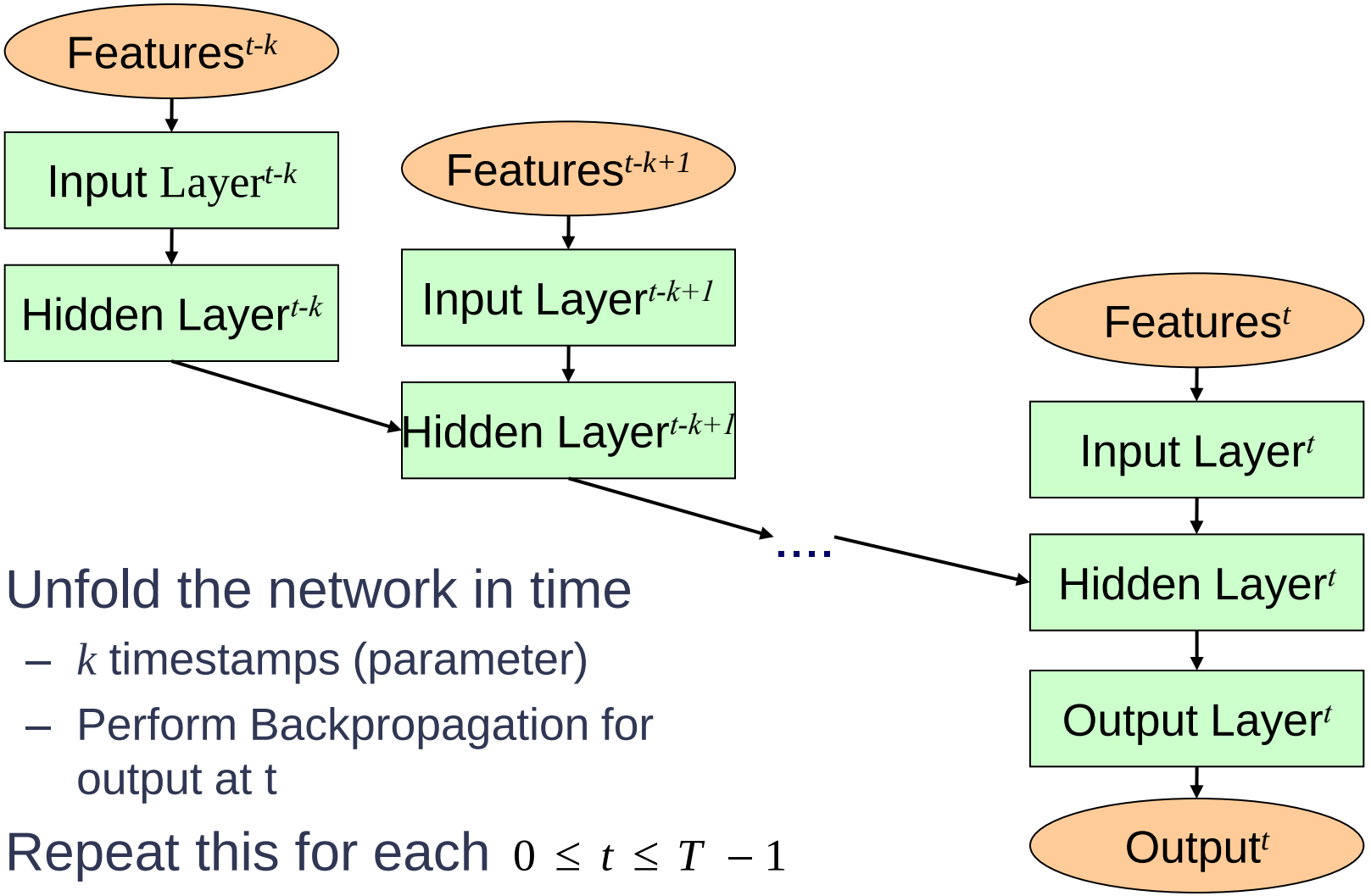


- Recurrent connections are added in order to keep information of previous time stamps in the network
- Novel equation for the activation:

$$a^t = \sum w_i x_i^t + \sum w_h b_h^{t-1}$$

- Context information is used
- How to train those networks …?

Training of RNNs – Backpropagation Through Time



- Unfold the network in time
  - k timestamps (parameter)
  - Perform Backpropagation for output at t
- Repeat this for each $0 \leq t \leq T - 1$

Sample Applications

- Sign language recognition[1]
  - 10 patterns, one person
- RNN with 1 hidden layer
  - 93 inputs
  - 150 hidden neurons
- Final recognition rate is 96%
- Already in 1991
  - Main problem: computational power (4 days for training)
- Later: 98% in real-time[2]

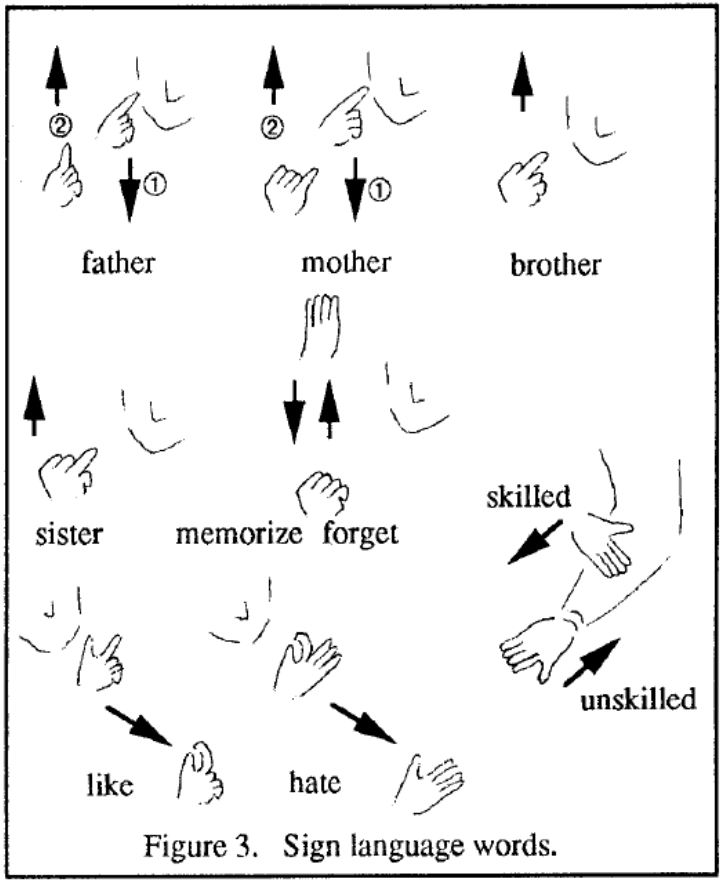


Figure 3. Sign language words.

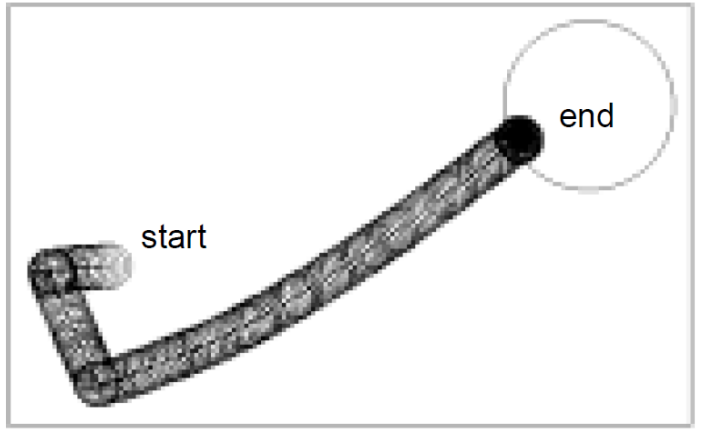
[1] Kouichi Murakami and Hitomi Taguchi. Gesture Recognition using Recurrent Neural Networks, 1991

[2] Thad Starner, Joshua Weaver, and Alex Pentland. Real-Time American Sign Language Recognition Using Desk and Wearable Computer Based Video, 1998 – Hidden Markov Model (HMM), 40 words in real-time, head-camera

Optimal Navigation in a Fixed Environment

- Adaptive Robot Behavior
  - Start anywhere and find target zone
  - Infrared proximity sensors + sensor for passing zone border (side must be remembered)
  - Rewards: fast, no wall, in target zone

- RNNs get about twice the points compared to MLP
  - Used fastest way after a maximum of 3 wall sights

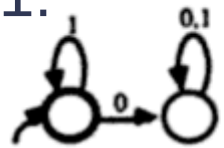


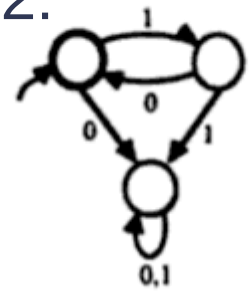
- Remark: genetic algorithm used
  - 100 individuals, 5,000 generations
  - 1% mutation rate (bit encoding for weights)

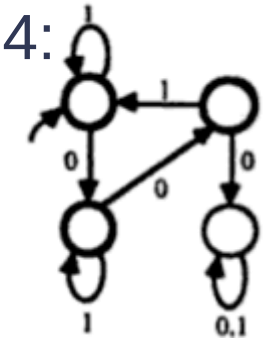
T. Ziemke. Remembering how to behave: Recurrent neural networks for adaptive robot behavior. Book Chapter 1999

Toy Problems Modeling Sequence Tasks

1. 2-sequence problem
   - Class 1: $1, a_1, a_2, a_3, a_4, \ldots, a_{T-1}$
   - Class 2: $-1, a_1, a_2, a_3, a_4, \ldots, a_{T-1}$
2. Parity problem
   - Is the number of 1 in a given Sequence (length T) even or odd?
3. Tomita grammars (7 exist, but 3 only considered)

1: 

2: 

4: 

(specifications)

T:length of sequence; a_i: any value from the interval $[-1,1]$

M. Tomita. Dynamic construction of finite automata from examples using hill-climbing, 1982

Two Big Guys in the Field of RNNs

Yoshua: Training RNNs with BPTT is difficult[1]

Jürgen: Learning RNNs for your problems is trivial[2]

Both: Gradient Descent is difficult but LSTM is good[3]

Yoshua Bengio, PhD 1991

Canada Research Chair in Statistical Learning Algorithms

Jürgen Schmidhuber, PhD 1991

Head of one of the world's top 10 AI labs, i.e., IDSIA in Switzerland

[1] Y. Bengio, P. Simard, and P. Frasconi. Learning Long-Term Dependencies with Gradient Descent is Difficult, IEEE Transactions on Neural Networks, VOL. 5, NO. 2, MARCH 1994

[2] S. Hochreiter, J. Schmidhuber. LSTM can Solve Hard Long Time Lag Problems, NIPS'9, 1997

[3] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. IEEE Press, 2001.

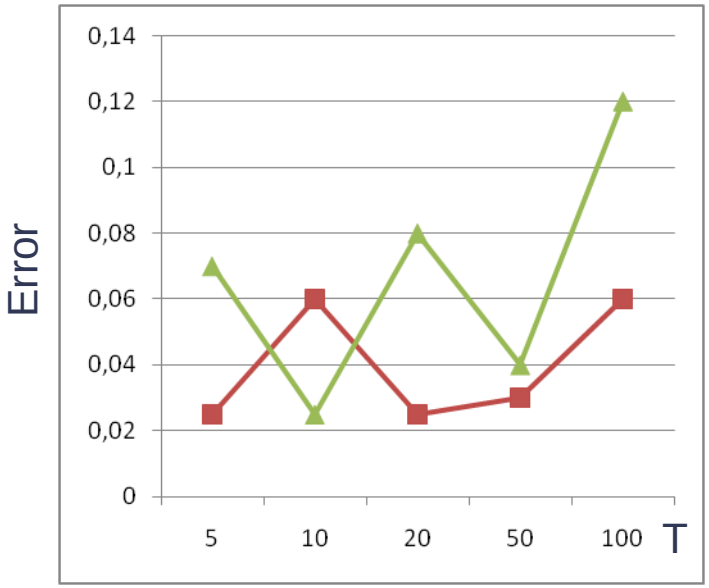
Experiments by Y. Bengio

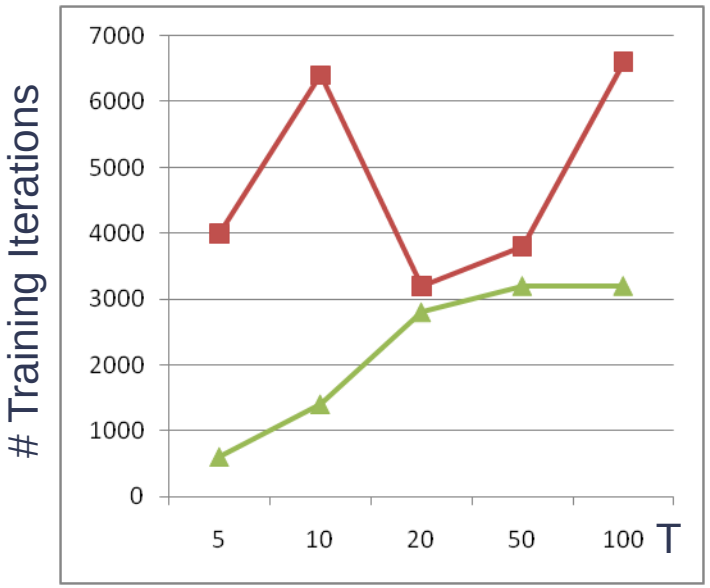
- Algorithms for solving problems: 2-sequence and parity
- Standard BPTT
- Simulated annealing
  - Initial distribution in the weight space
  - Generate new points randomly based on temperature C
  - Always keep the best neuron(s) and reduce C after every few steps
- Multi-grid random search (similar to simulated annealing)
  - Only use new points when error is reduced
- Pseudo-Newton optimization
- Time-weighted pseudo-Newton optimization
- Discrete error propagation

2-Sequence Problem

- Class 1: $1, a_1, a_2, a_3, a_4, \ldots, a_{T-1}$
- Class 2: $-1, a_1, a_2, a_3, a_4, \ldots, a_{T-1}$

- multi-grid random search
- simulated annealing

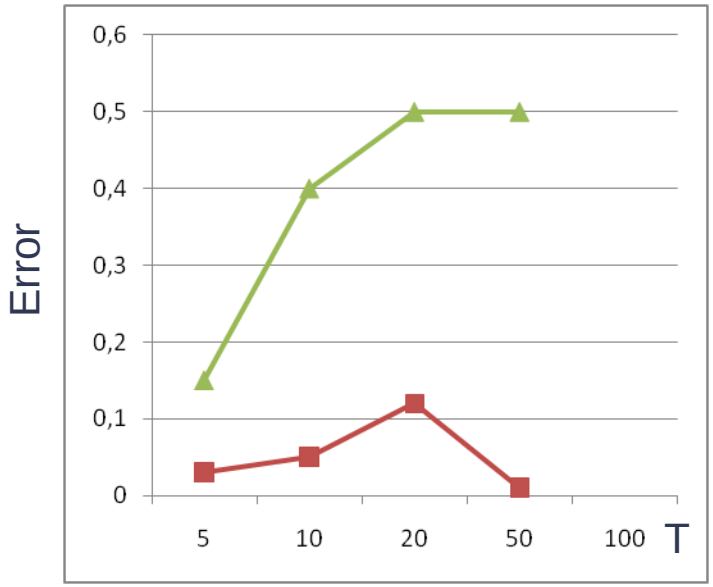


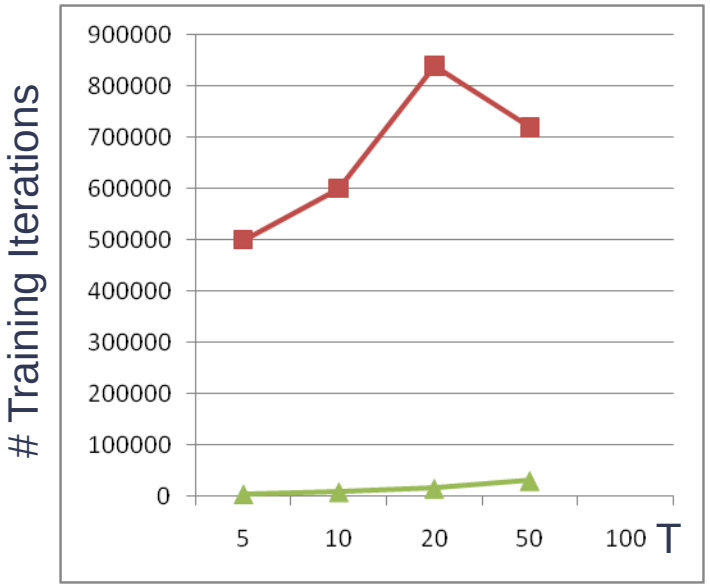


Parity problem

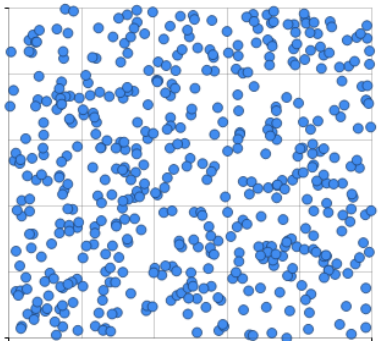
- Class 1 example: *1, 0, 0, 1, 1*
- Class 2 example: *1, 1, 0, 0, 1, 1*

- multi-grid random search
- simulated annealing

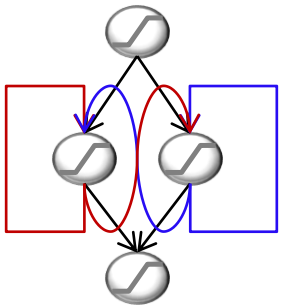


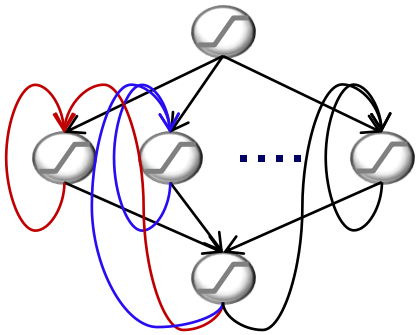


Idea by J. Schmidhuber



- Random search in the weight space
  - Initialize weights randomly in [-100,100]
  - Repeat until success
- Two architectures
  - Inputs are -1 (for 0) and 1 (for 1), Targets are 1 or 0
  1. Fully connected net
     - 1 input, one output
     - n hidden units
  2. Same as 1. but
     - n=10
     - Each hidden unit only recurrently connected with itself and output





- Search stopped when training error below 0.1
- Trivial Task: can be solved quickly by random search

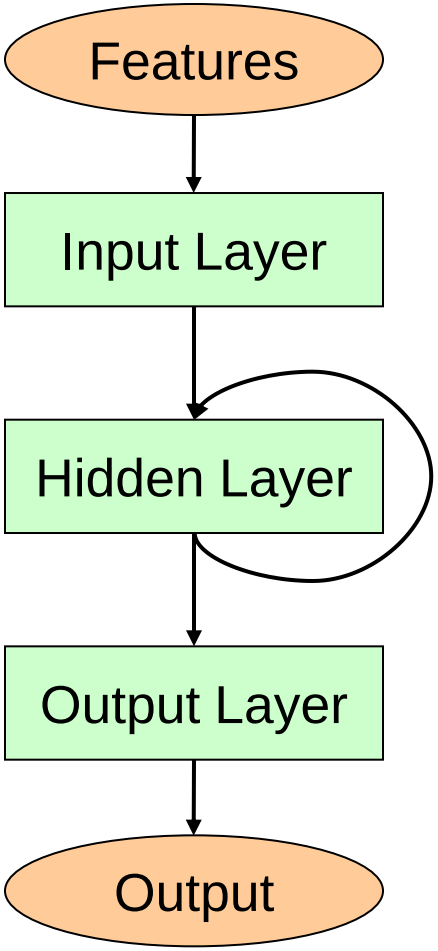
Experiments

Problem	Best algorithm		RS, Architecture 1		RS, Architecture 2	
	Best error	Trials	Best error	Trials	Best error	Trials
2-seq	Multi-grid random search		n=1			
	0.06	6,400	<0.001	1,247	<0.001	718
Parity	Simulated annealing		n=1			
	0	810,000	0	2,906	0	2,797
Tomita	Other methods		n=1,3,2 (for G 1, 2, 4)			
G 1	0	23,000	0	182	0	288
G 2	0	77,000	0	1,511	0	17,953
G 4	0	46,000	0	13,833	0	35,610

100 training sequences (50 per class), 100 test sequences, T = 500-600 (harder)

Example of a Non-Trivial Task

- Adding Problem (slightly modified)
  - Sequence elements are pairs (a_i, b_i)
  - Values of b_i is 0 or 1
  - Aim: sum up over all pairs where b_i is 1
  - Example 1: (0.2, 0), (0.5, 0), (0.3, 1), (0.6, 0), (-0.9, 1), … (-0.3, 0)
- Sequence generation
  - Only two samples contain pairs where b_i is 1
  - First one x_1 is in first ten pairs
  - Second one x_2 is anywhere else in first half of the sequence
  - Target result is: 0.5 + (x_1+x_2) /2, error should be less than 0.04
- Unable to solve with known RNN learning algorithms within reasonable time



- Recurrent connections are added in order to keep information of previous time stamps in the network
- Novel equation for activation:

$$a^t = \sum w_i x_i^t + \sum w_h b_h^{t-1}$$

- Can be written in matrix form

$$A^t = W_i \cdot X^t + W_h \cdot B^{t-1}$$

- Context information is used, however: impossible to store precise information over long durations

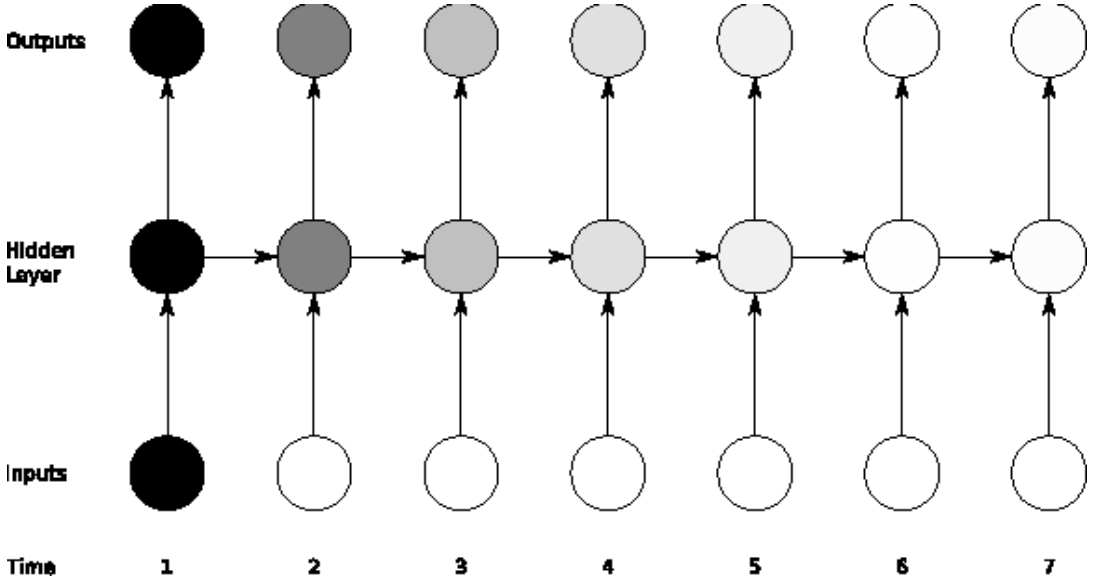
- Usual RNN forget information after a short period of time

$$A^t = W_i \cdot X^t + W_h \cdot B^{t-1} = W_i \cdot X^t + W_h \cdot h(W_i \cdot X^{t-1} + W_h \cdot B^{t-2})$$

- Example:
  Neuron

During
  7 timestamps

- Information
  vanished



- Usual RNN forget information after a short period of time

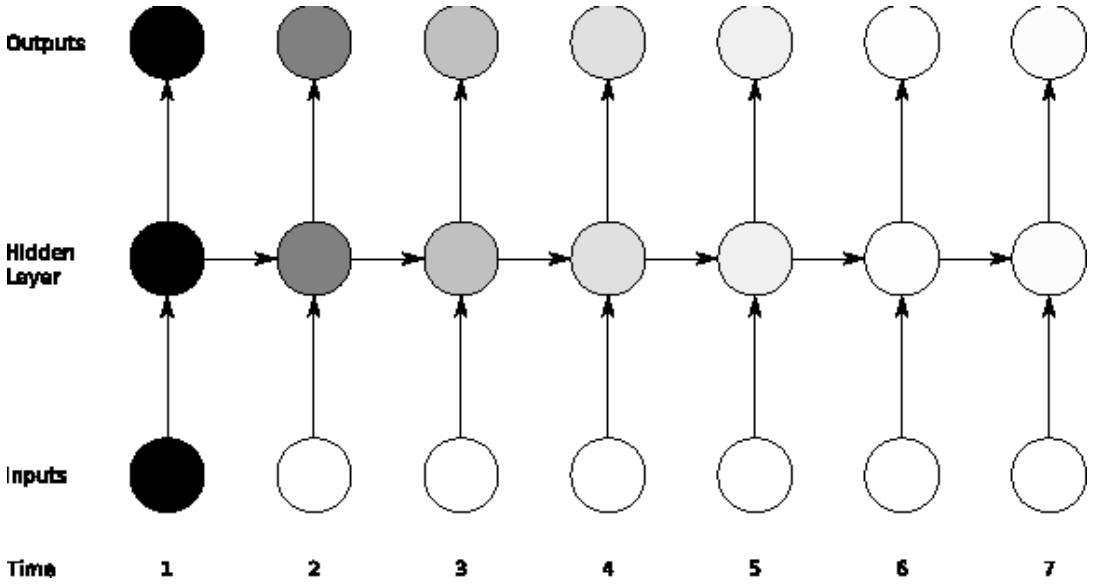
$$A^t = W_i \cdot X^t + W_h \cdot B^{t-1} = W_i \cdot X^t + W_h \cdot h(W_i \cdot X^{t-1} + W_h \cdot B^{t-2})$$

$$\Rightarrow A^t = f(X^t, W_h X^{t-1}, W_h{}^2 X^{t-2}, \ldots, W_h{}^t X^1)$$

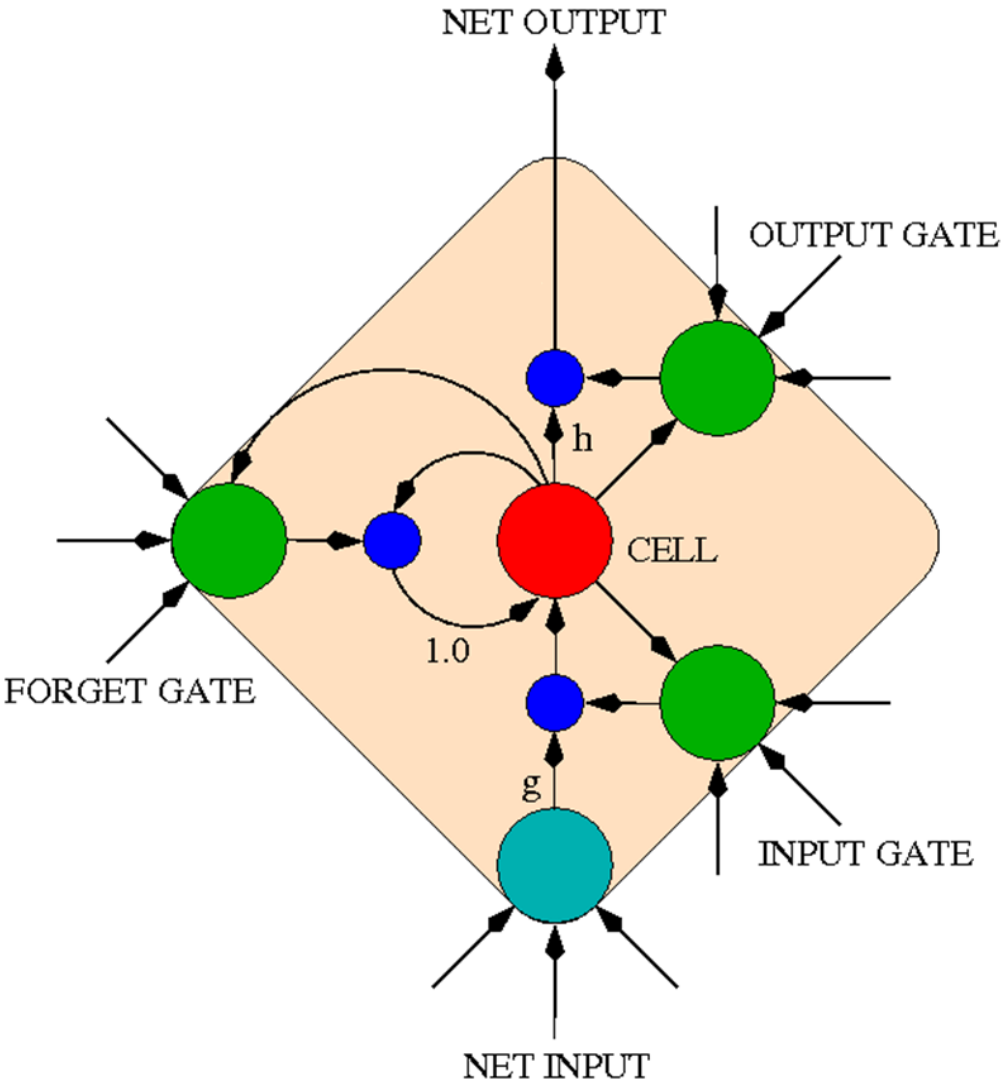
- Example:
  Neuron

During
  7 timestamps

- Information
  vanished



Core Idea: New Memory Cell Instead of Perceptron



- Memory cell
  - Read, write and reset operations
- Input Gate (single cell)

$$a_\iota^t = W_{a,i} \cdot X^t + W_{h,i} \cdot B^{t-1} + W_{c,i} S_c^{t-1}$$

- Forget Gate

$$a_\theta^t = W_{h,\theta} \cdot X^t + W_{h,\theta} \cdot B^{t-1} + W_{c,\theta} S_c^{t-1}$$

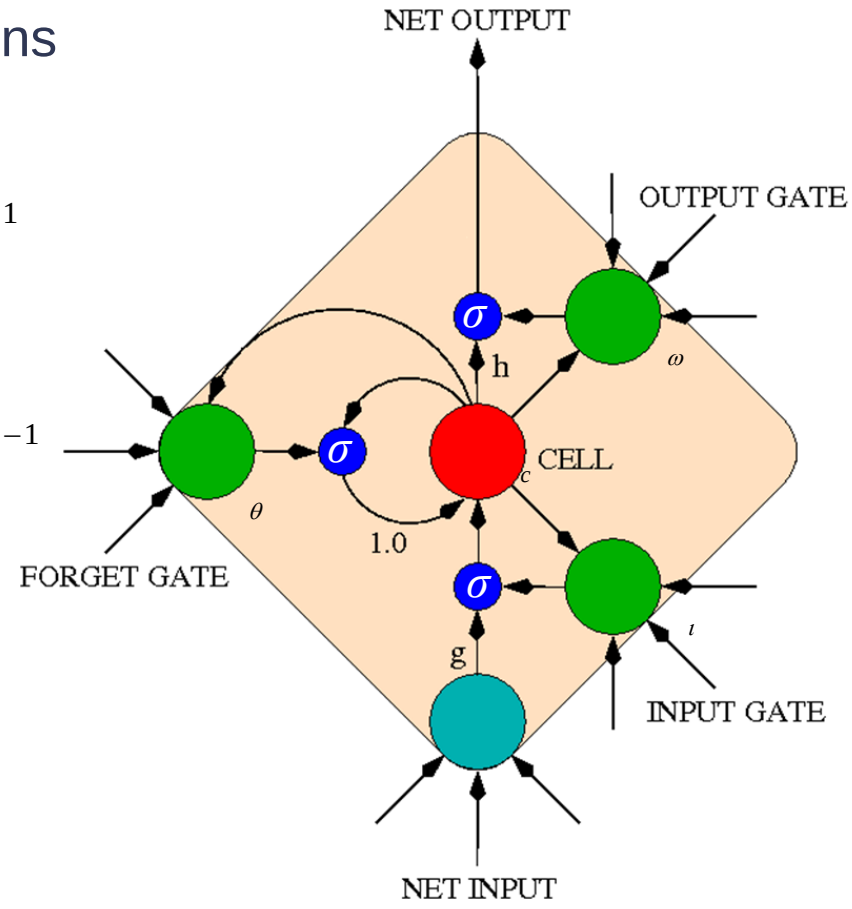
- Cell State

$$a_c^t = W_{a,c} \cdot X^t + W_{h,c} \cdot B^{t-1}$$

$$s_c^t = \sigma(a_i^t)\, g(a_c^t) + \sigma(a_\theta^t)\, s_c^{t-1}$$

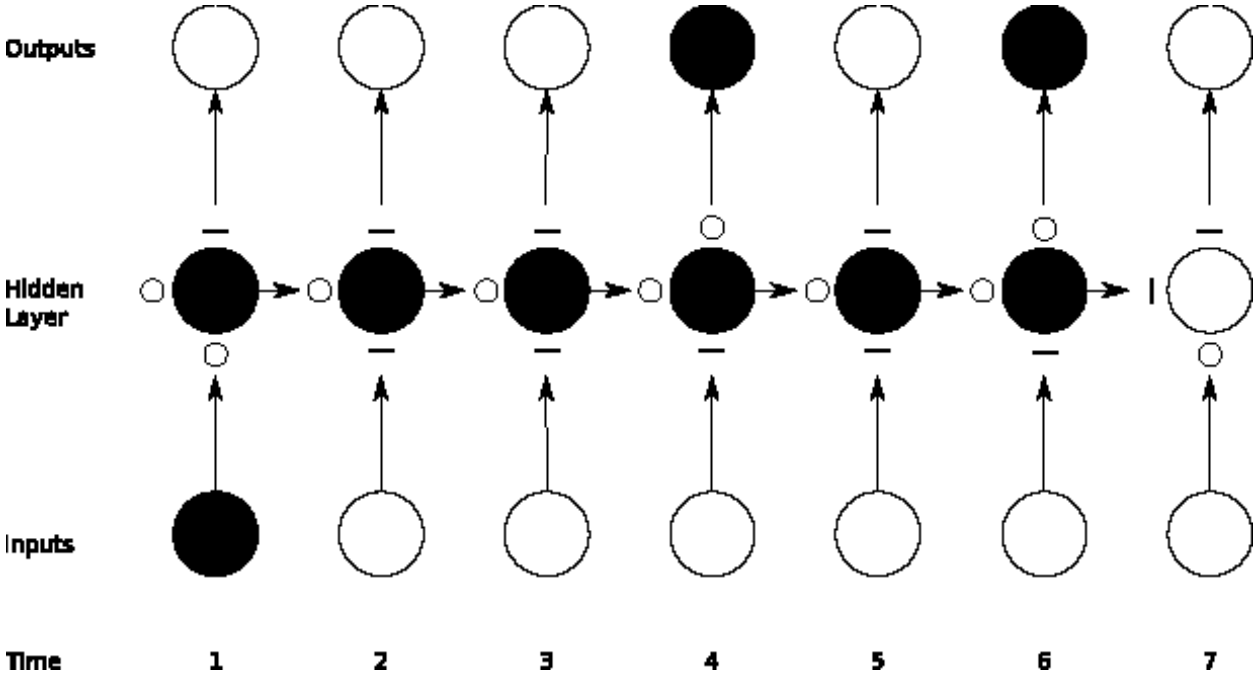
- Assume σ is close to 0 or 1

$$\Rightarrow\ s_c^t = [0\ or\ 1]\, g(a_c^t) + [0\ or\ 1]\, s_c^{t-1}$$

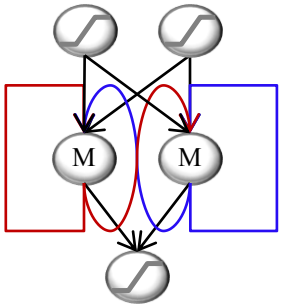


No Vanishing Gradient

- Output Gate $a_{\omega}^{t} = W_{a,\omega} \cdot X^{t} + W_{h,\omega} \cdot B^{t-1} + W_{c,\omega} S_{c}^{t}$
- Output $b_{c}^{t} = \sigma(a_{\omega}^{t}) h(s_{c}^{t})$
- Neuron now
  - O : open ($\sigma=1$)
  - | : closed ($\sigma=0$)



LSTM Applications

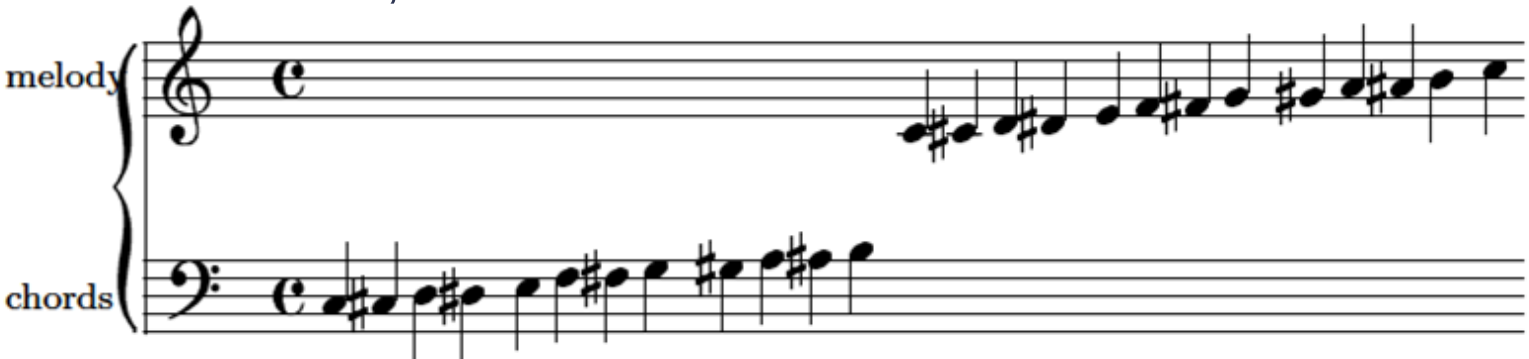


- Adding Problem
- Architecture
  - 3 layers (2 input, 1 output, 2 LSTM)
  - g and h are sigmoid
- Training
  - Random sequences used
  - Stopped when error was below 0.01 for the last 2,000 sequences
- Results (average of 10 trials)
  - Test set: 2,560 sequences

	T=100	T=500	T=1,000
# training iterations	74,000	209,000	853,000
# of (error > 0.01)	1	0	1

Music: Blues Improvisation

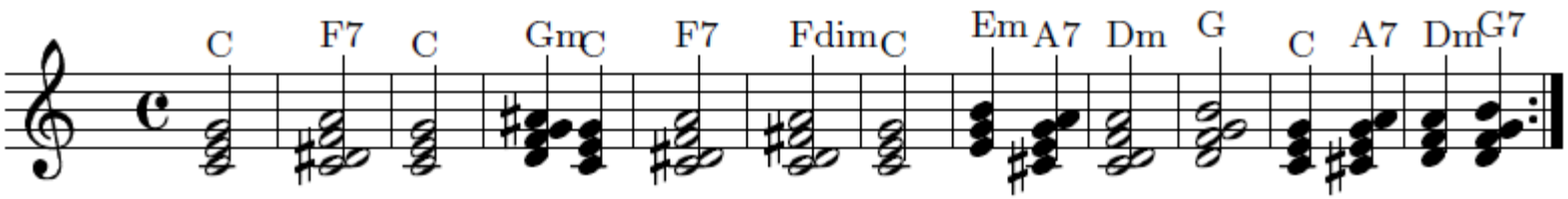
- First attempts in 1989[1]
  - RNNs for a note-by-note composition
  - Mozer: „While the local contours made sense, the pieces were not musically coherent, lacking thematic structure and having minimal phrase structure and rhytmic organization
- LSTM in 2002[2]
  - 13 melody notes, 12 chord notes (25 inputs)
  - -1 means off, 1 means on



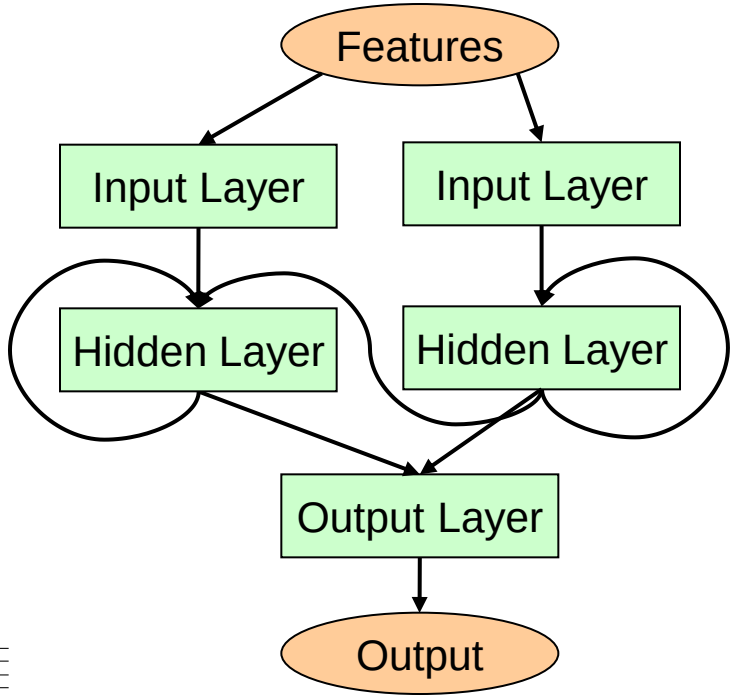
[1] M. C. Mozer, “Neural network composition by prediction: Exploring the benefits of psychophysical constraints and multi-scale processing,” Cognitive Science, vol. 6, pp. 247-280, 1994.

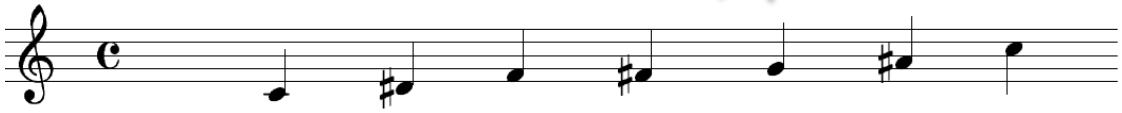
[2] Eck, D.; Schmidhuber, J. .Finding temporal structure in music: blues improvisation with LSTM recurrent networks, 2002

- Bebop music
  - 12 bar blues
  - 8 notes per bar (no shorter than 1/8th notes)
- Experiment 1
  - Only chords were presented
  - Training sequence length 96
  - Test input: initially 3 bars and then the output of the net is used
  - Network could perfectly learn the chords after 15 minutes on a 1GHz Pentium



- Experiment 2
  - 4 LSTM for chords
  - 4 LSTM for melody
  - Only self-recurrent and from chords to melody LSTM
  - Pentatonic notes from music pieces used for training
  - Only quarter notes





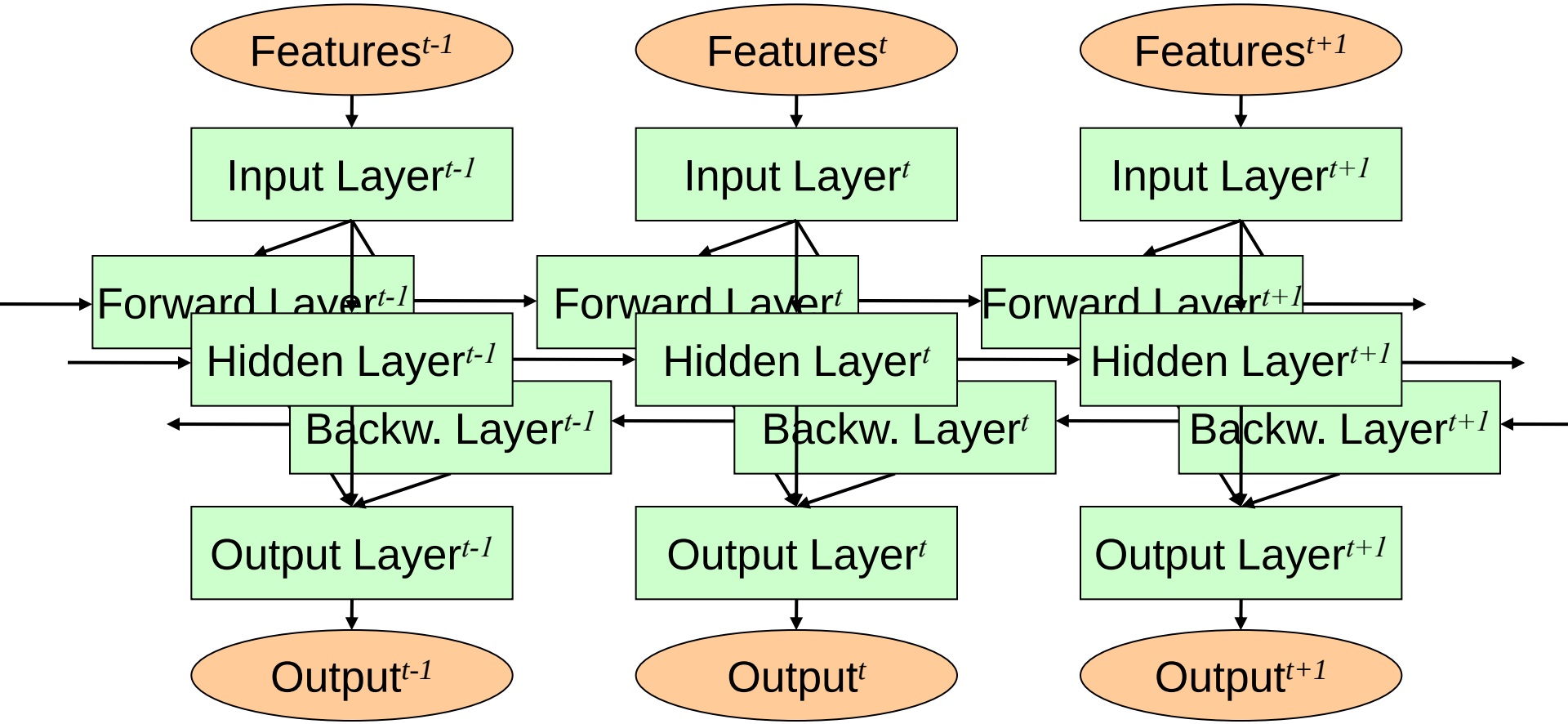
- Results
  - Freely improvisated
  - Samples: random lstm 1 lstm 2

LSTM is not Enough

- Several relevant, but more difficult problems exist
  - Complete sequence recognition
  - Sequence to sequence matching
- Applications
  - Speech recognition
  - Handwriting recognition
  - Protein localization
- Often the context from later is also interesting[1]
  - Long or short ([a] or [a:])
  - Idea: Delayed output – but how long?
  - Better idea: use context from whole sequence

[1] Mike Schuster and Kuldip K. Paliwal, Bidirectional Recurrent Neural Networks, IEEE TSP, 1997

Bidirectional RNN



- Trained with backpropagation through time (forward path through all time stamps for each hidden layer sequentially)

Frame-Wise Phoneme Classification

- TIMIT database
  - Texas Instruments and Massachusetts Institute of Technology
  - 3,696 training phonemes
  - 1,344 testing phonemes
  - Speaker independent
- Experiments with several comparable architectures
  - 26 input units (MFCC features)
  - 61 output units (one for each phoneme)
  - All networks had roughly the same number of weights (100,000)
  - Examples: BLSTM 93, LSTM 140, BRNN 185, RNN 275

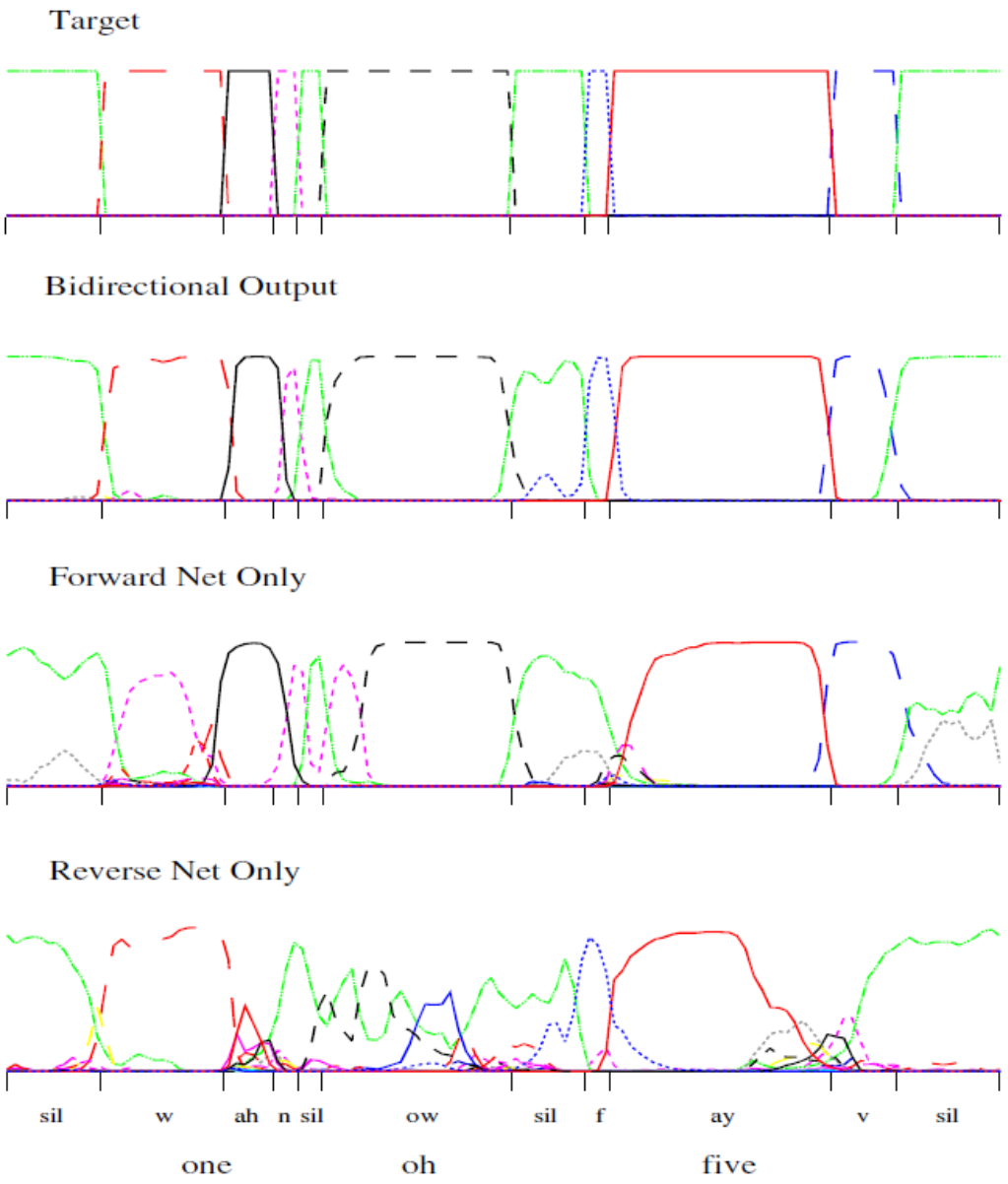
[1] Alex Graves, Jürgen Schmidhuber. Framewise phoneme classification with bidirectional LSTM and other neural network architectures, Neural Networks, 2005

Results

Network	Training set (%)	Test set (%)	Epochs
BLSTM (retrained)	78.6	70.2	17
BLSTM	77.4	69.8	20.1
BRNN	76.0	69.0	170
BLSTM (Weighted Error)	75.7	68.9	15
LSTM (5 frame delay)	77.6	66.0	34
RNN (3 frame delay)	71.0	65.2	139
LSTM (backwards, 0 frame delay)	71.1	64.7	15
LSTM (0 frame delay)	70.9	64.6	15
RNN (0 frame delay)	69.9	64.5	120
MLP (10 frame time-window)	67.6	63.1	990
MLP (no time-window)	53.6	51.4	835
RNN (Chen and Jamieson, 1996)	69.9	74.2	–
RNN (Robinson, 1994; Schuster, 1999)	70.6	65.3	–
BRNN (Schuster, 1999)	72.1	65.1	–

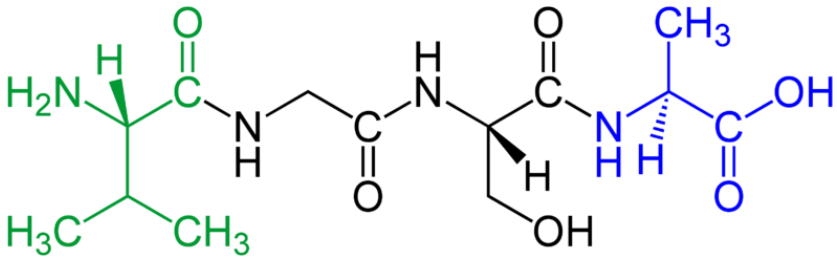
Retraining is done by increasing the target delay after each 5 epochs

Closer Look into the Behaviour



Support Human Experts Processing lots of Data

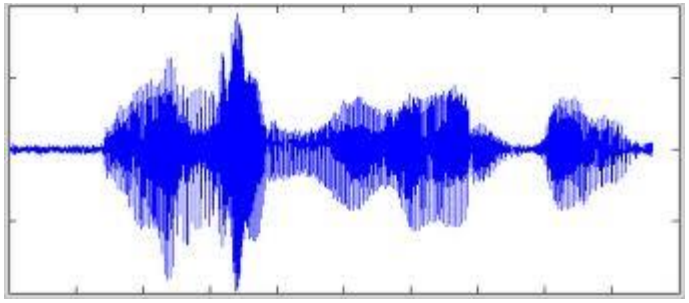
- Task: Localize Proteins
  - Given a sequence of N-Terminal Residues (T=70)
  - Classify the type of novel protein (3 classes)
- Architecture
  - 3 outputs
  - 1 input
  - 3 LSTM in forward and 2 LSTM in backward layer
- Result:
  - MLP: 90.0 %
  - BRNN: 91.3 %
  - BLSTM: 93.3 %



[1] Trias Thireou, Martin Reczko. Bidirectional Long Short-Term Memory Networks for Predicting the Subcellular Localization of Eukaryotic Proteins

Improvements of BLSTM

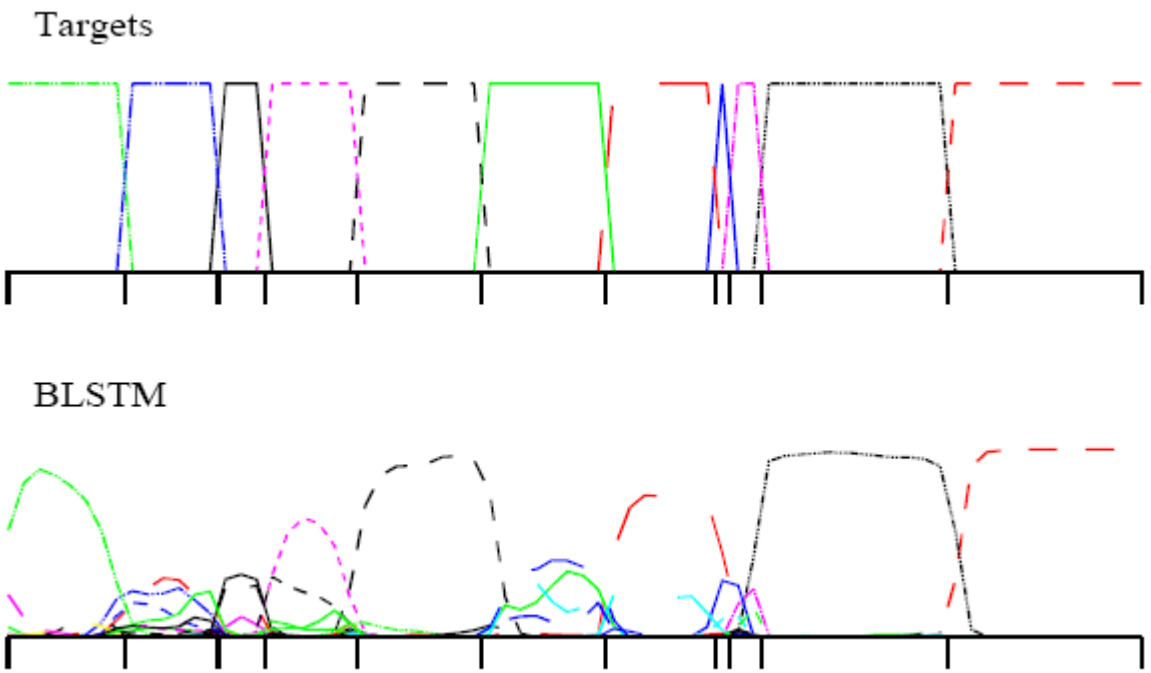
- I will now present you the last few missing ingredients to build a powerful sequence-to-sequence matching, especially for handwriting and speech recognition
- Input: raw features, raw pixel data, or raw point-sequence
- Output machine-readable transcription



→ “The sound of”

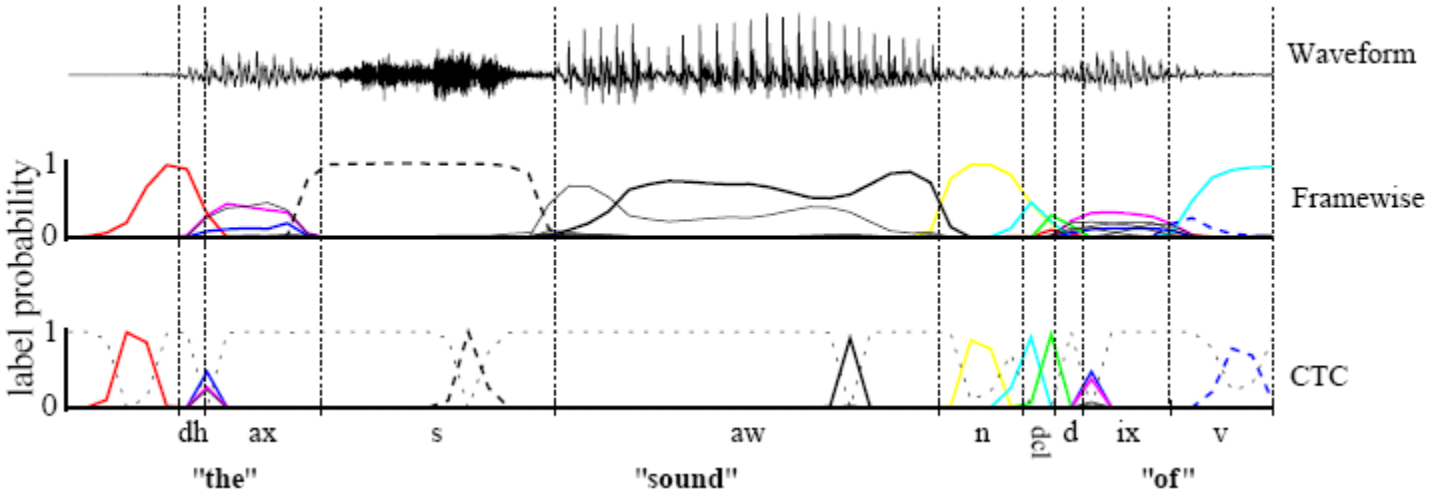
- Easy to use
- Based on LSTM

Standard Framewise Classification

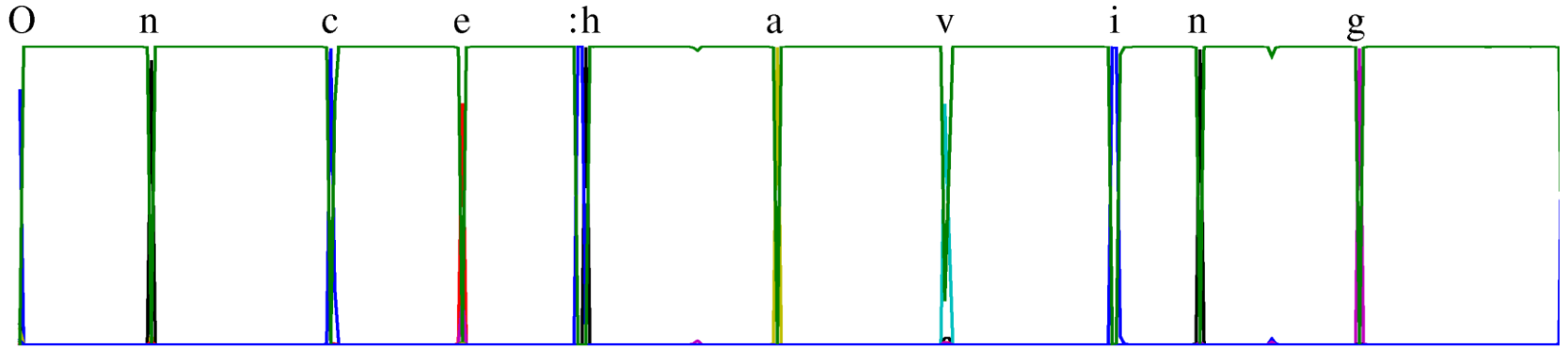


- Segmentation is needed
- Information of previous and next frames is not available
- Idea: introduce a way of connected temporal classification

- Connected Temporal Classification (CTC)
- Example for speech:



- Additional blank label (*b* green)
- Allows application to whole sequences
- Output with normalized likelihood for each word



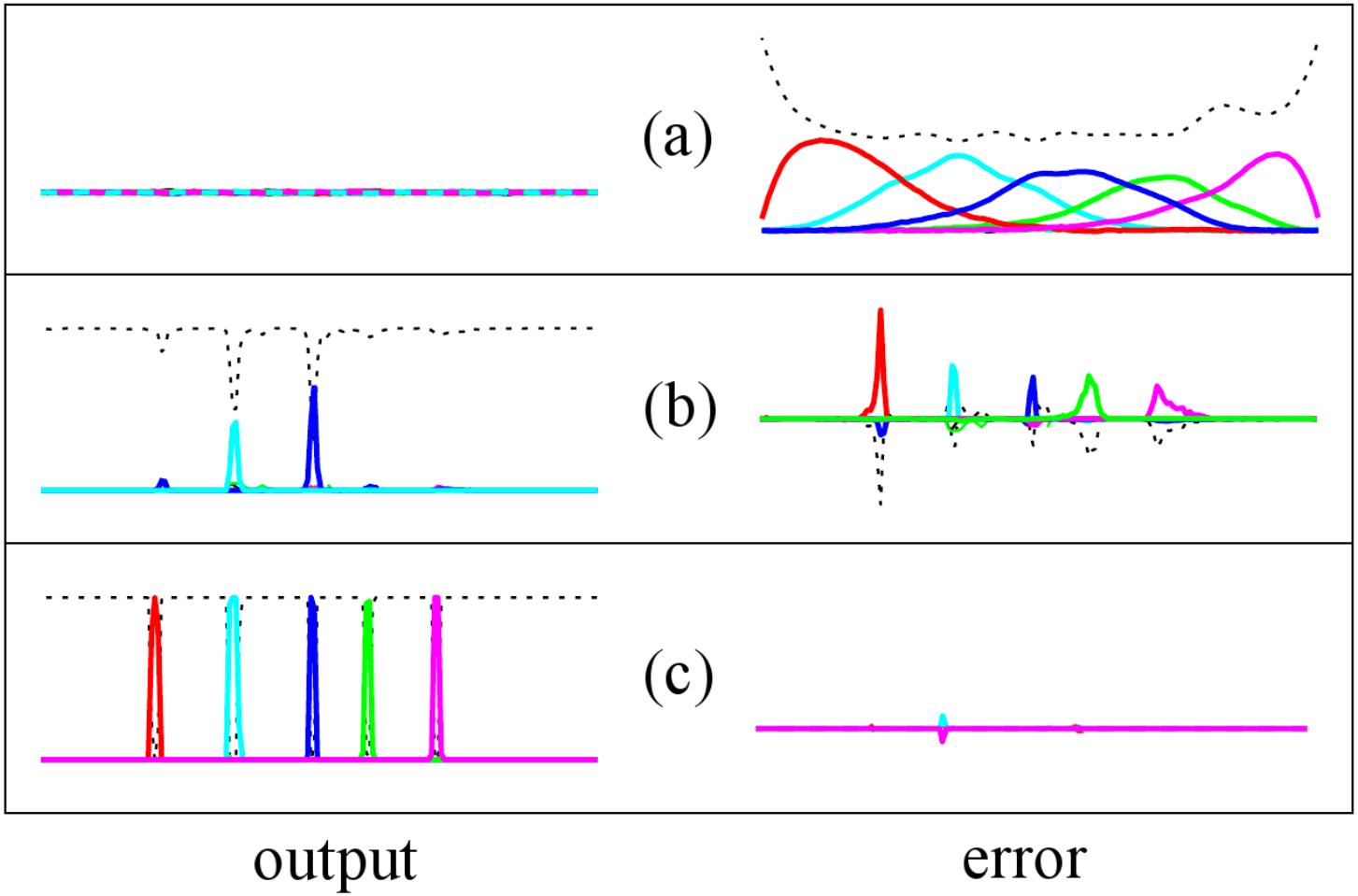
- Training: objective function is smoothed and recalculated after each iteration (details in references)
- Testing: similar to Viterbi-algorithm

How Does it Behave? Training Error Example

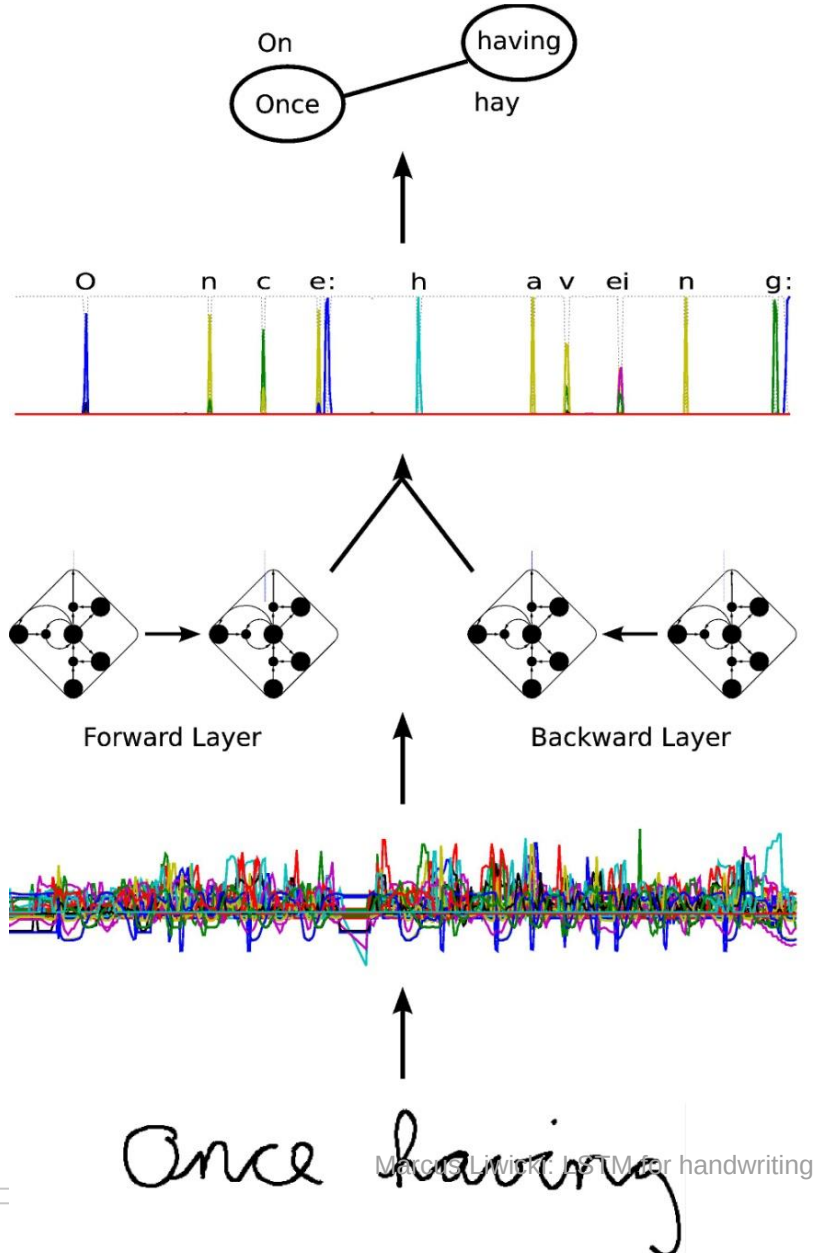
Beginning (random)

10 iterations (error aroud predictions)

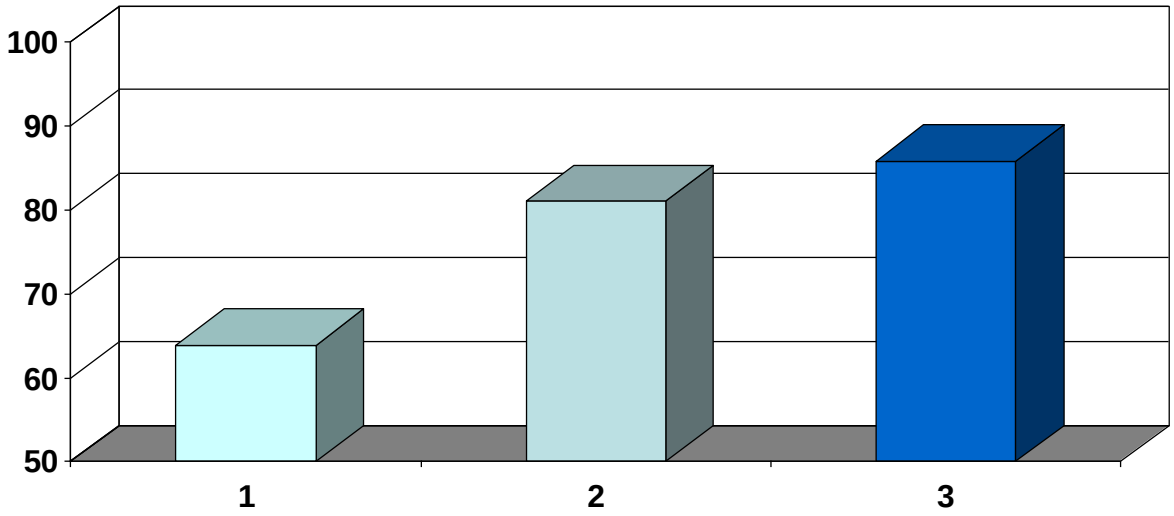
Final (nearly no error)



Overall System



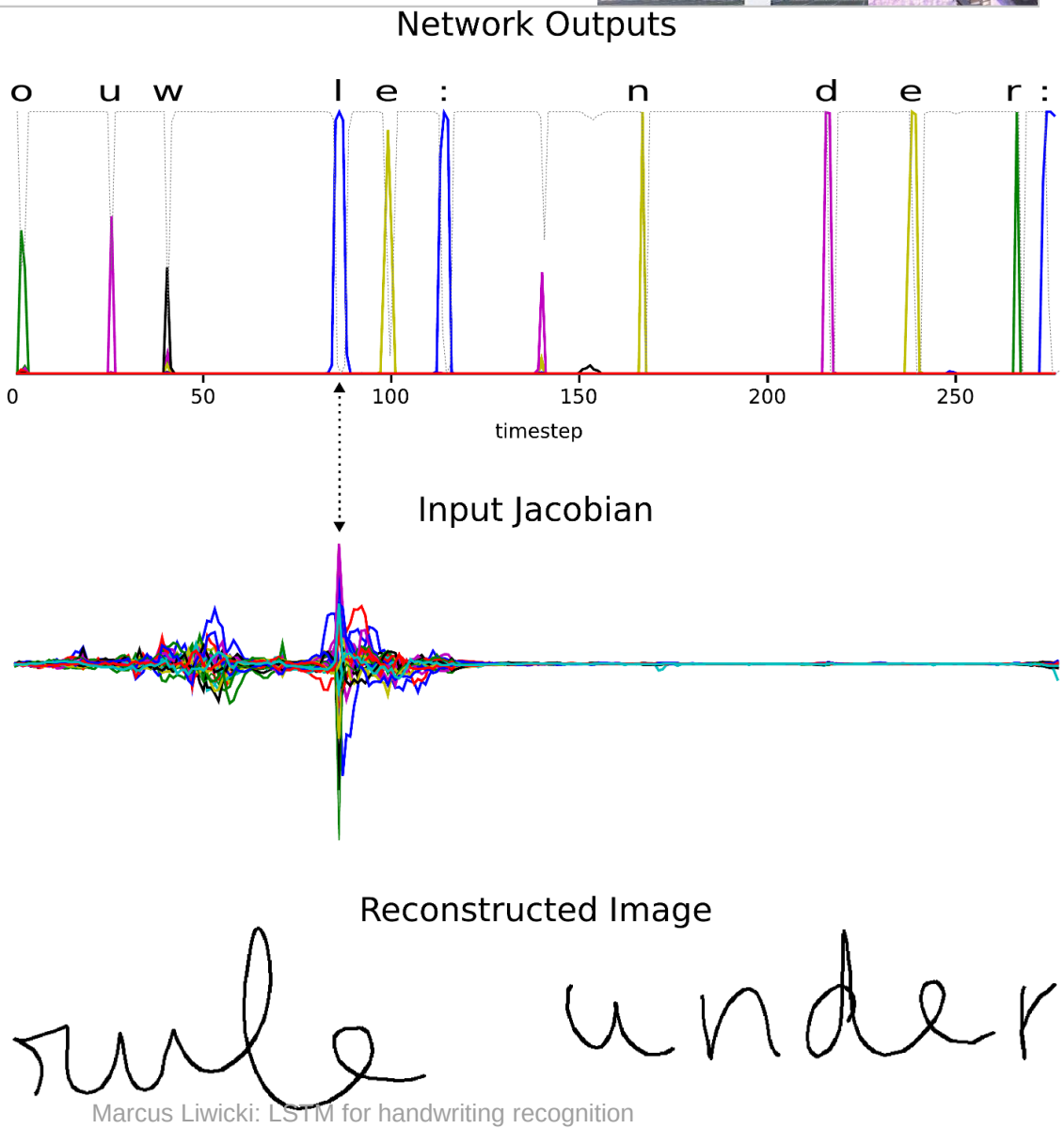
Handwriting Recognition Experiments



- Experiments
  1. 63.86% with Hidden Markov Model (HMM)
  2. 81.05% with BLSTM (100 cells) and CTC
  3. 86 % after combination of several classifiers

Information Preservation Experiment

- Example
  - Output at l
  - Amount of information for each cell derived from each time stamp
  - Called input Jacobian
  - Estimated by Backprop.



Discussion: BLSTM vs. Conventional HMM

- BLSTM are discriminative
- BLSTM allow correlated input features
- Internal states are continuous and multivariate, because they are defined by the vector of activations of the hidden units
- The output is a sequence of labels without duration information
- BLSTM is in principle able to access context from the entire input sequence

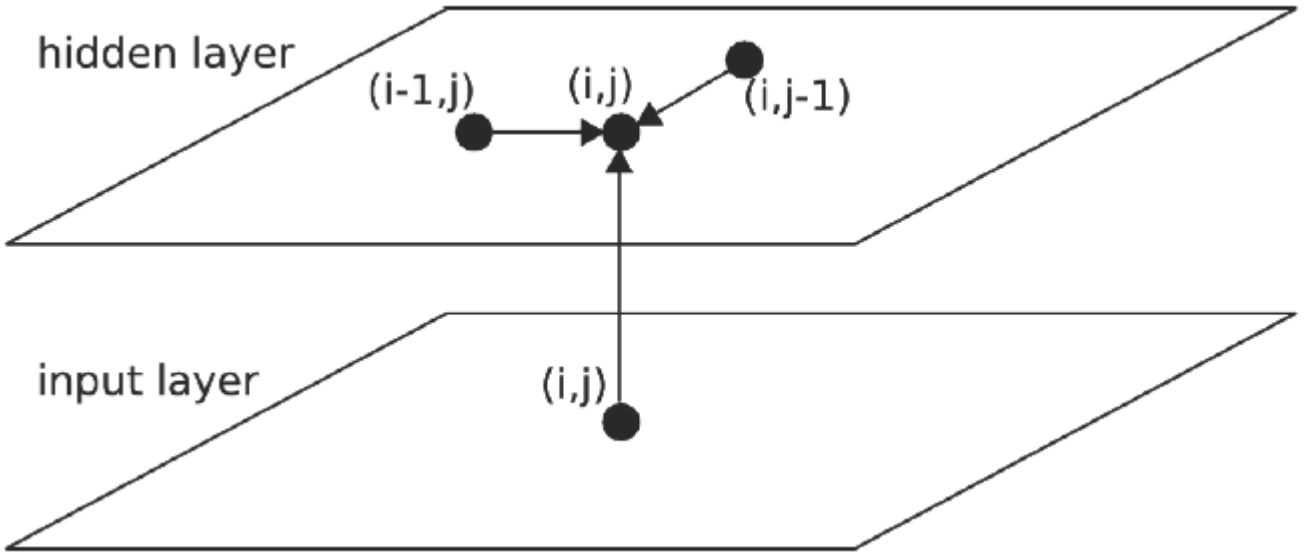
Going Into Multiple Dimensions

- If input size is fixed, MLP can be applied, but what if size is unknown?
- Face recognition

- Idea: Sliding window in multiple directions

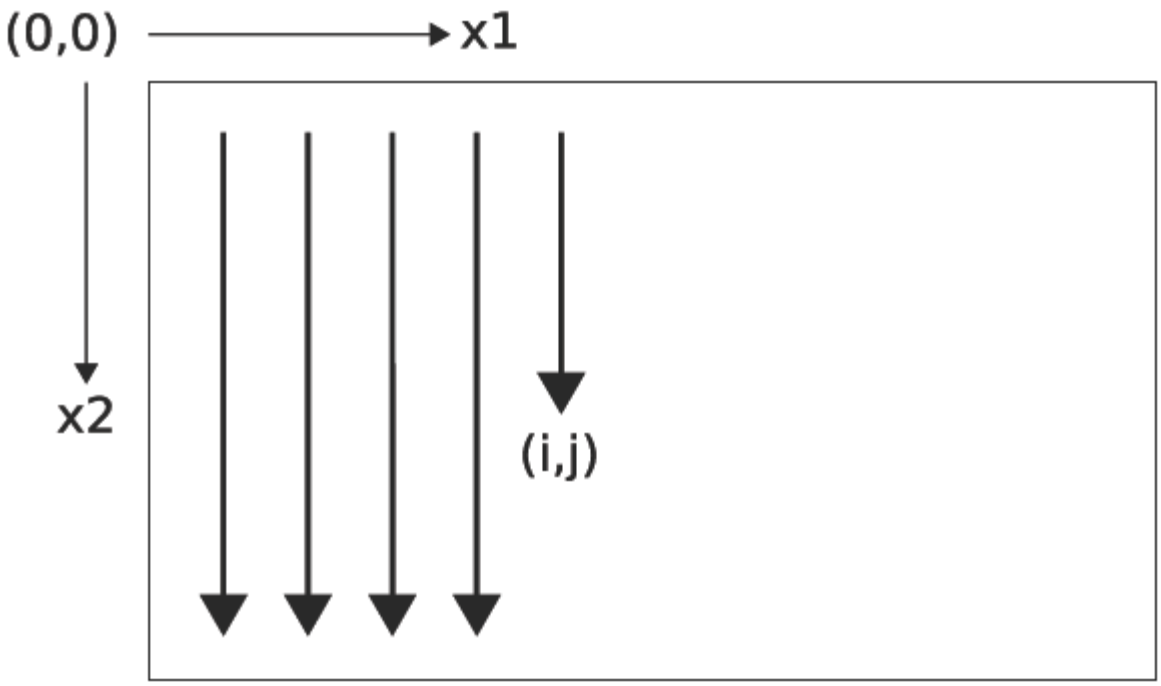
Going Into Multiple Dimensions

- Concrete idea (like DAG-RNN):
  - Each neuron receives external input and its own activation from one step back along all dimensions
  - Can be applied to any dimensional sequences (img 2D, video 3D)



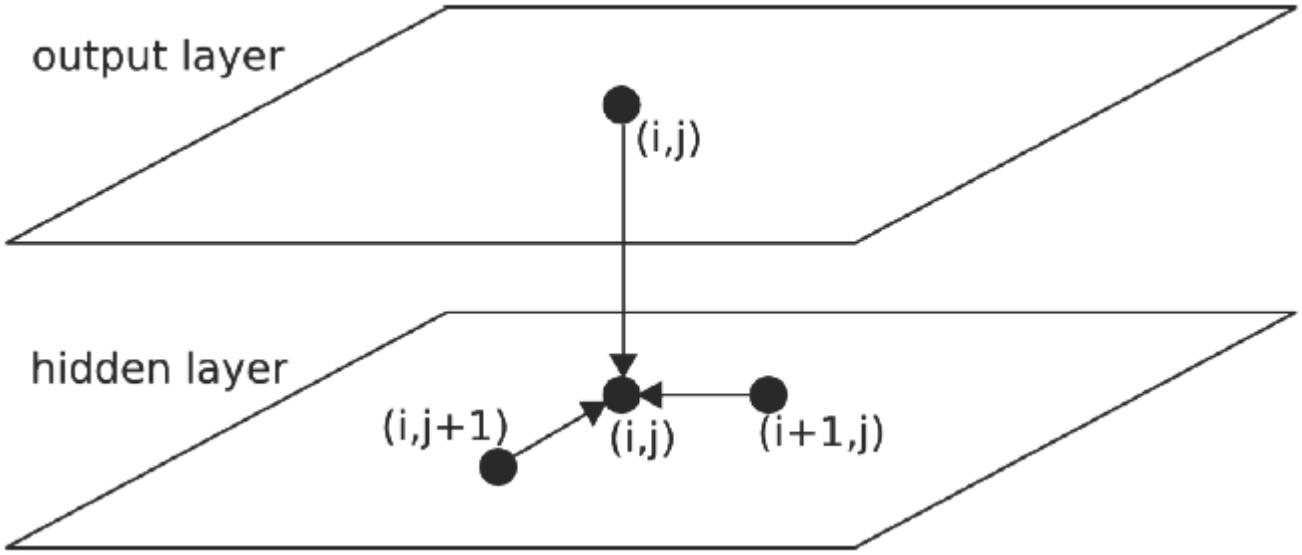
Sequence Ordering Through Forward Pass

- Ensure that each previous' step output is already calculated
- Example:

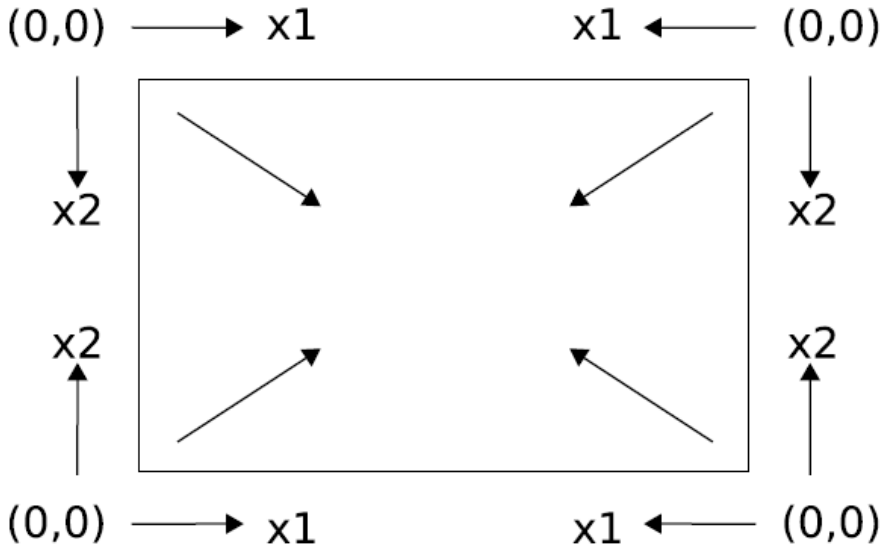


- Note: Boundaries have to be omitted

- N-dimensional backpropagation through time



- Each neuron receives external input and its own activation from one step back along all dimensions
- Can be applied to any dimensional sequences (2D image, 3D movie, 4D with time, 6D for robot control)
- Idea: Use 2^D hidden layers

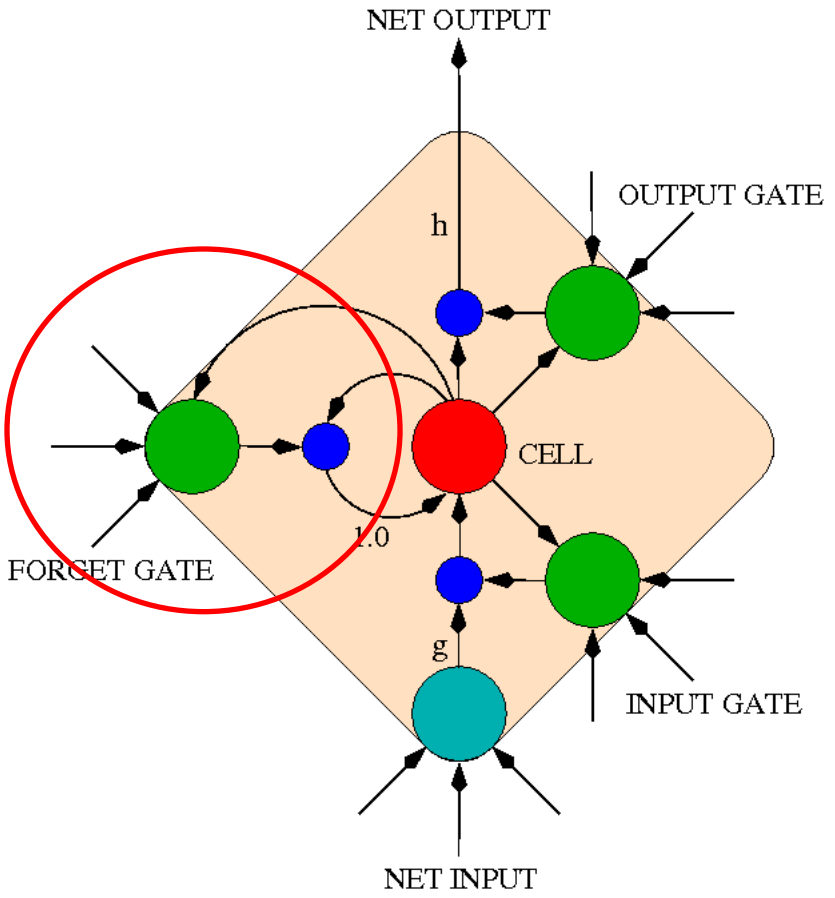


Problem

- Does the complexity explode?
- 2^d seems to be quite large
- However
  - Number of weights has more influence
  - Several calculations can be shared
- Furthermore
  - Reduce the size of the hidden layers with increasing dimensionality
  - It has been found that for speech recognition the number of weights was reduced to half and MDRNN gave still better results
- Main scaling concern is the size of the data, i.e., the length of the sequence

Combining idea with BLSTM

- Introduce 2^d self-connections, i.e., 2^d forget gates, each connected along one dimension
- However, only one input gate (connected to all dimensions)
- Also, only one output gate, since only the cell state is considered



Overall system

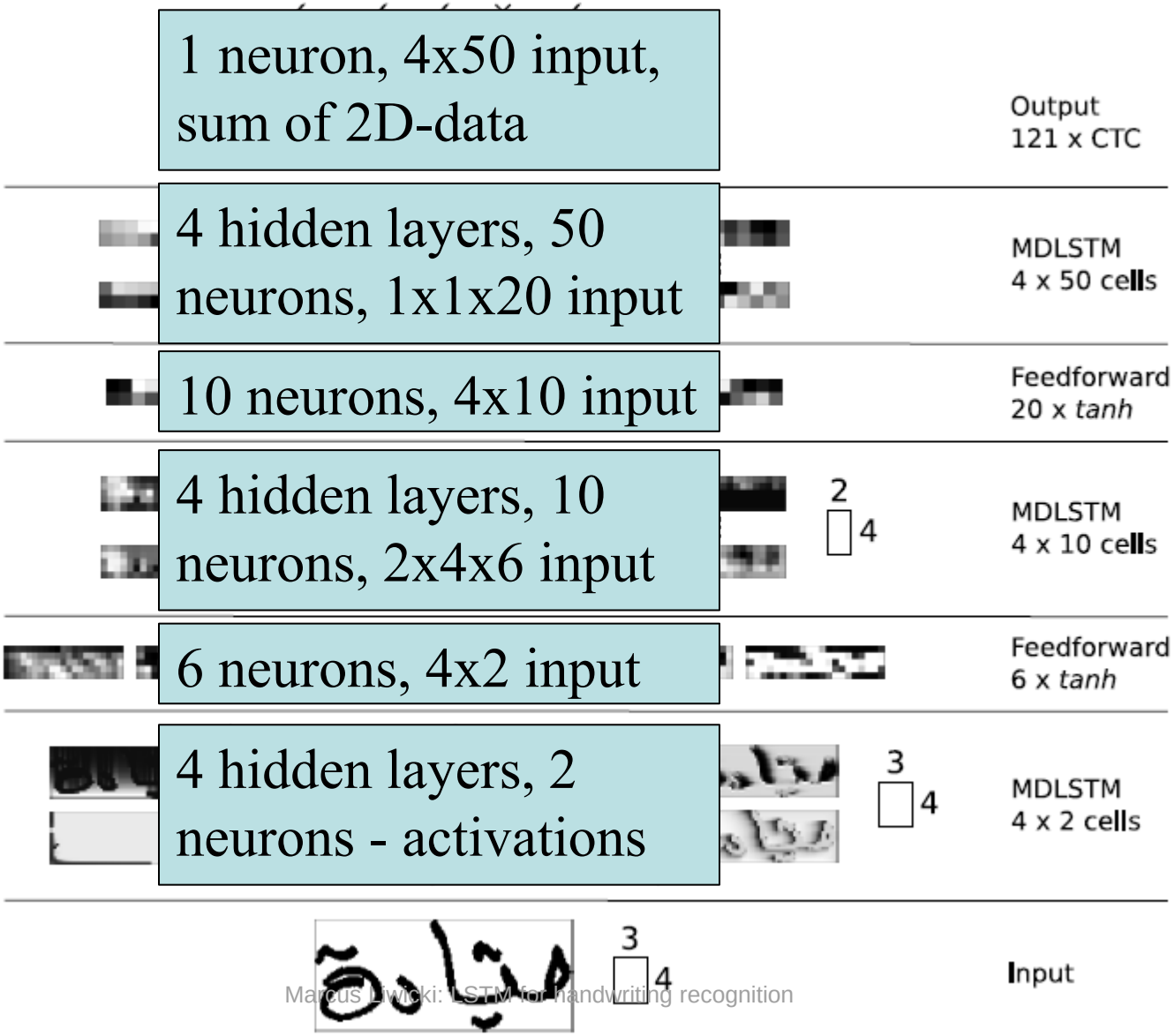
MDLSTM layers and feed-forward layers

4x3=>12-dim vector

Small at bottom large at top

159,369 weights but most at top

1D at top by summing up



ICDAR 2007 Arabic handwriting recognition contest

System	SET f			SET s		
	top 1	top 5	top 10	top 1	top 5	top 10
CACI-3	14.28	29.88	37.91	10.68	21.74	30.20
CACI-2	15.79	21.34	22.33	14.24	19.39	20.53
CEDAR	59.01	78.76	83.70	41.32	61.98	69.87
MITRE	61.70	81.61	85.69	49.91	70.50	76.48
UOB-ENST-1	79.10	87.69	90.21	64.97	78.39	82.20
PARIS V	80.18	91.09	92.98	64.38	78.12	82.13
ICRA	81.47	90.07	92.15	72.22	82.84	86.27
UOB-ENST-2	81.65	90.81	92.35	69.61	83.79	85.89
UOB-ENST-4	81.81	88.71	90.40	70.57	79.85	83.34
UOB-ENST-3	81.93	91.20	92.76	69.93	84.11	87.03
SIEMENS-1	82.77	92.37	93.92	68.09	81.70	85.19
MIE	83.34	91.67	93.48	68.40	80.93	83.73
SIEMENS-2	87.22	94.05	95.42	73.94	85.44	88.18
MDLSTM	**91.43**	**96.12**	**96.75**	**78.83**	**88.00**	**91.05**

A. Graves and J. Schmidhuber. Offline handwriting recognition with multidimensional recurrent neural networks. In Advances in Neural Information Processing Systems 21, 2009.

Results of the ICDAR 2009 Arabic HWR Contest

System	Word Accuracy	Time/Image
CTC	81.06%	371.61 *ms*
Arab-Reader HMM	76.66%	2583.64 *ms*
Multi-Stream HMM	74.51%	143,269.81 *ms*

4 Summarized results from the offline Arabic handwriting recognition competition

Volker Märgner, Haikal El Abed, "ICDAR 2009 Arabic Handwriting Recognition Competition," Document Analysis and Recognition, International Conference on, pp. 1383-1387, 2009 10th International Conference on Document Analysis and Recognition, 2009

Results of the ICDAR 2009 French HWR Contest

System	Word Accuracy
CTC	93.17%
HMM+MLP Combination	83.17%
Non-Symmetric HMM	76.34%

Summarized results from the offline (French) handwriting recognition competition

Grosicki, E.; El Abed, H.; , "ICDAR 2009 Handwriting Recognition Competition," *Document Analysis and Recognition, 2009. ICDAR '09. 10th International Conference on* , vol., no., pp.1398-1402, 26-29 July 2009

And the Very Best

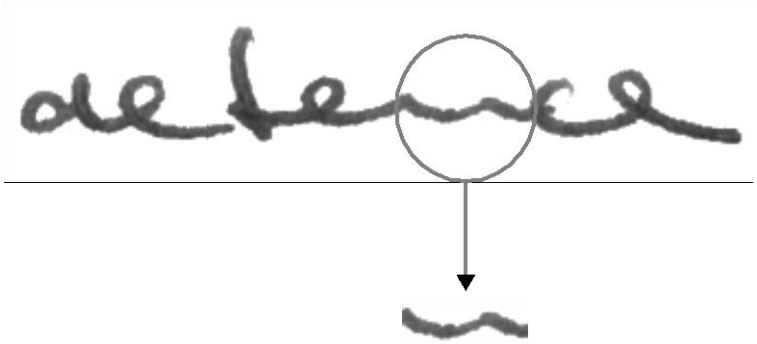
- It is open source and for free
- sourceforge.net/projects/rnnl/
- Examples are online (Arabic recognition)
- More later!

Other Application

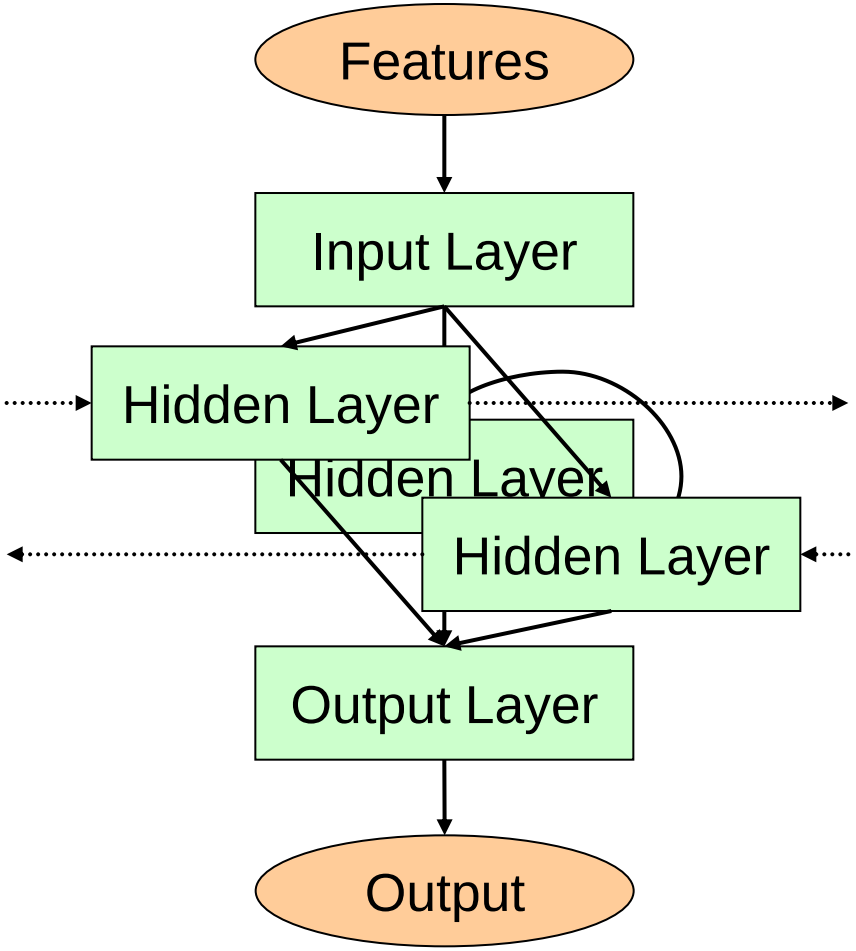
- Associative memory
- Encoding/decoding information
- Real-Time learning

Conclusion: BLSTM

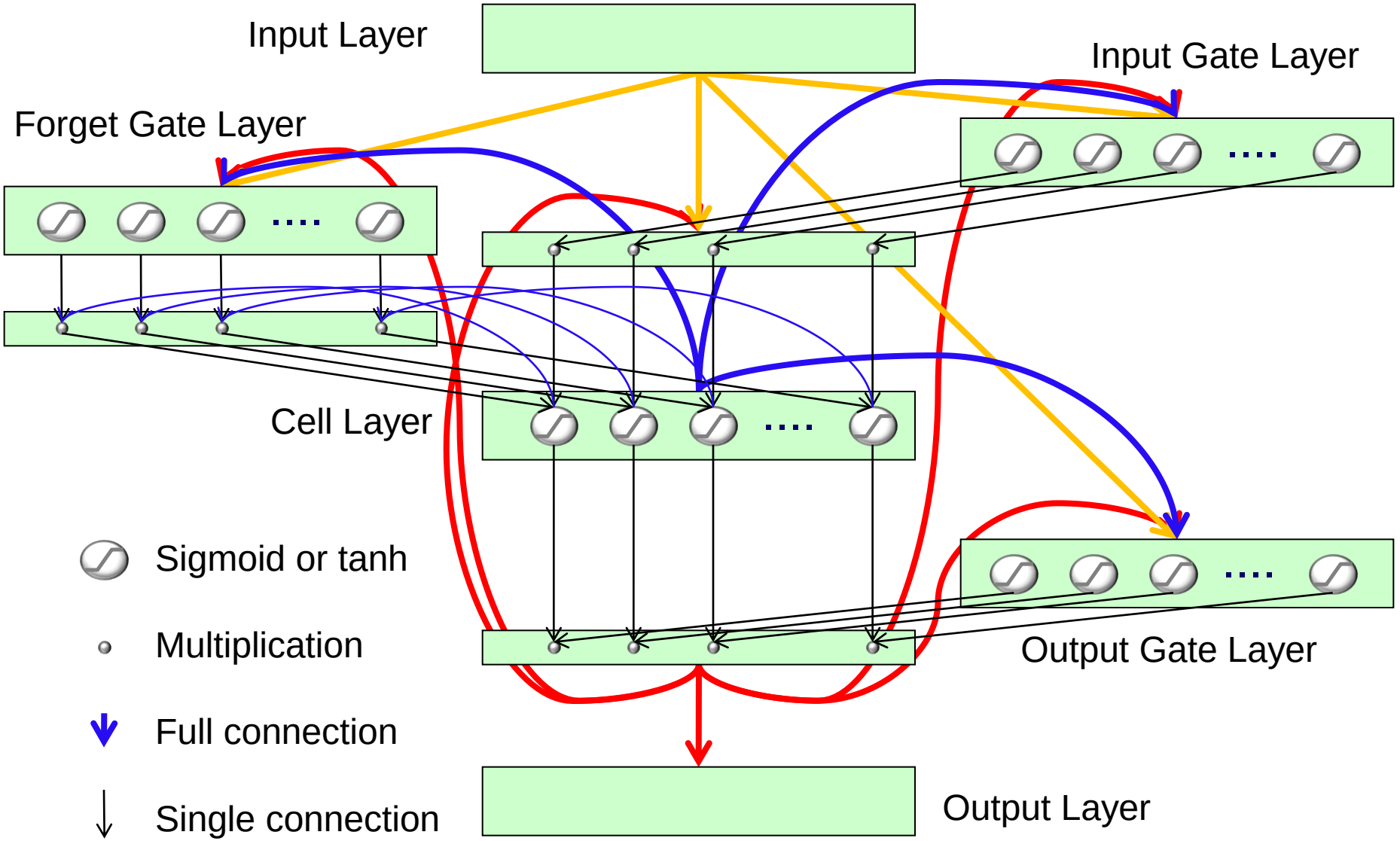
- Importance of context



- Multilayer perceptron network
- Recurrent connections
- Bidirectional
- Memory instead of perceptron



Discussion: RNN - LSTM



Conclusion

- Neural networks simulate human neurons to
  - Recognize symbols (MLP)
  - Recognize sequential information (RNN)
  - Store precise information over long durations (LSTM)
  - Access information of complete sequence (BLSTM)
  - Map sequences of different length (CTC)
  - Process multi-dimensional sequences (MDLSTM)
- During the years they became more powerful
  - Better architectures and algorithms
  - Faster hardware
- Diverse application areas

Do-It-Yourself – Toolkits and how to apply them

http://www.ecmlab.de/jannlab

http://pybrain.org/

sourceforge.net/projects/rnnl/

Java Implementation

- JANNLab
  - Written by Sebastian Otte, Master Student supervised by me
  - Currently still in beta-stadium
  - Will soon be available at: http://ecm.mi.hs-rm.de/jannlab
- http://code.google.com/p/touch-and-write/source/
  - There is a version available
  - /touchandwrite-pr-toolbox/src/main/java/de/dfki/touchandwrite/rnn/
  - Added during the revision dd084b5218f3
  - Can be easily downloaded and used for testing
- I will now go through a test case
  - The code will be available for download

Example Implementation

- import ...Net;
- import ...core.CellType;
- import ...data.Sample;
- import ...data.SampleSet;
- import ...data.SampleTools;
- import ...generator.LSTMGenerator;
- import ...generator.MLPGenerator;
- import ...generator.NetCoreGenerator;
- import ...io.Serializer;
- import ...learning.GradientDescent;
- import ...learning.LearningTask;
- import ...learning.tools.TrainerTools;

Main method

```
Random rnd = new Random(System.currentTimeMillis());
```

- First, define the single layers with Macro classes:

```
NetCoreGenerator gen = new NetCoreGenerator();
int input  = MLPGenerator.inputLayer(gen, 2);
int hidden = LSTMGenerator.lstmLayer(
        gen,
        2,                  // number of blocks.
        CellType.SIGMOID,   // gates activation.
        CellType.TANH,      // netinput activation (g).
        CellType.TANH,      // state activation (h).
        true                // peepholes?);
int output = MLPGenerator.outputLayer(gen,  1, CellType.TANH);
```

- Next, connect them fully:

```
gen.weightedLinkLayer(input, hidden);
gen.weightedLinkLayer(hidden, hidden);
gen.weightedLinkLayer(hidden, output);


Net net = gen.generate();
System.out.println(net);
```

- Load the data sets

```
SampleSet trainset =
    SampleTools.readCSV("experiments/adding/trainset_50_10000.c
    sv");
int trainlength    = trainset.maxSequenceLength();
SampleSet testset  = SampleTools.
      readCSV("experiments/adding/testset_50_5000.csv");
int testlength     = testset.maxSequenceLength();
Sstem.out.println("trainset samples       : " + trainset.size());
System.out.println("trainset max.seqlength : " + trainlength);
System.out.println("testset samples        : " + testset.size());
System.out.println("testset max.seqlength  : " + testlength);
System.out.println();
```

- Initialize

```
net.rebuffer(Math.max(trainlength, testlength));
net.initializeWeights(rnd);
```

- Generate a Gradient Trainer and fill it with the parameters

```
GradientDescent trainer = new GradientDescent();
trainer.setNet(net);
trainer.setRnd(rnd);
trainer.setTrainset(trainset);
trainer.setValidationset(testset);
trainer.setTask(LearningTask.CLASSIFICATION);
trainer.setTargetError(0.0);
trainer.setLearningRate(0.001);
trainer.setMomentum(0.9);
trainer.setEpochs(30);
trainer.setValidationInterval(2);
trainer.setValidationAbort(5);
```

- Train

  trainer.train();

- Save

  Serializer.write(net, "experiments/adding/net_50.gz");

- Test

```
double thres = 0.04;
int ok = 0;
for (Sample s : testset) {
    net.reset();
    double err = TrainerTools.performForward(net, s,
      LearningTask.CLASSIFICATION);
    if (err < thres) {
      ok++;}}
double ratio = 0.0;
if (ok > 0) {
    ratio = 100.0 * ((double)ok) / ((double)testset.size());}
System.out.println("testset result: " + ratio + "%.");
```

Other Toolkits

- Pybrain
  - Based on python
  - http://pybrain.org/
  - Nice tutorials at: http://pybrain.org/docs/
- RNNLib
  - Based on C++
  - sourceforge.net/projects/rnnl/
  - A bit of a hack to make it running, but it is quite fast

Last Slide

Thank You!

Dr. Marcus Eichenberger-Liwicki

TU Kaiserslautern & DFKI, Germany

Marcus.Liwicki@dfki.de