Deep Neural Networks (Głębokie Sieci Neuronowe)

Module 3: Recurrent Neural Networks

Krzysztof Krawiec

Wydział Informatyki i Telekomunikacji
Politechnika Poznańska
2019/2020
http://www.cs.put.poznan.pl/kkrawiec/

Outline

Recurrent Architectures

Recurrent vs. recursive

recurrent | rɪˈkʌr(ə)nt |

adjective

- occurring often or repeatedly: she had a recurrent dream about falling.

recursive | rɪˈkəːsɪv |

adjective

characterized by recurrence or repetition.

- *Mathematics & Linguistics* relating to or involving the <u>repeated application of a rule, definition, or procedure to successive results</u>: this restriction ensures that the grammar is recursive.
- *Computing* relating to or involving a program or routine of which <u>a part requires the application of the whole</u>, so that its explicit interpretation requires in general many successive executions: a recursive subroutine.

Learning from sequences

Sequences:

- $x^0, x^1, \ldots, x^t, \ldots.$
- In general vectors

Key properties:

- Structure present along the time axis
    - No permutation invariance.
- Variable length

Where it all started (more or less)

Long short-term memory

Sepp Hochreiter; Jürgen Schmidhuber

Neural Computation, 9 (8), 1997: 1735–1780.

Main challenges in learning time sequences

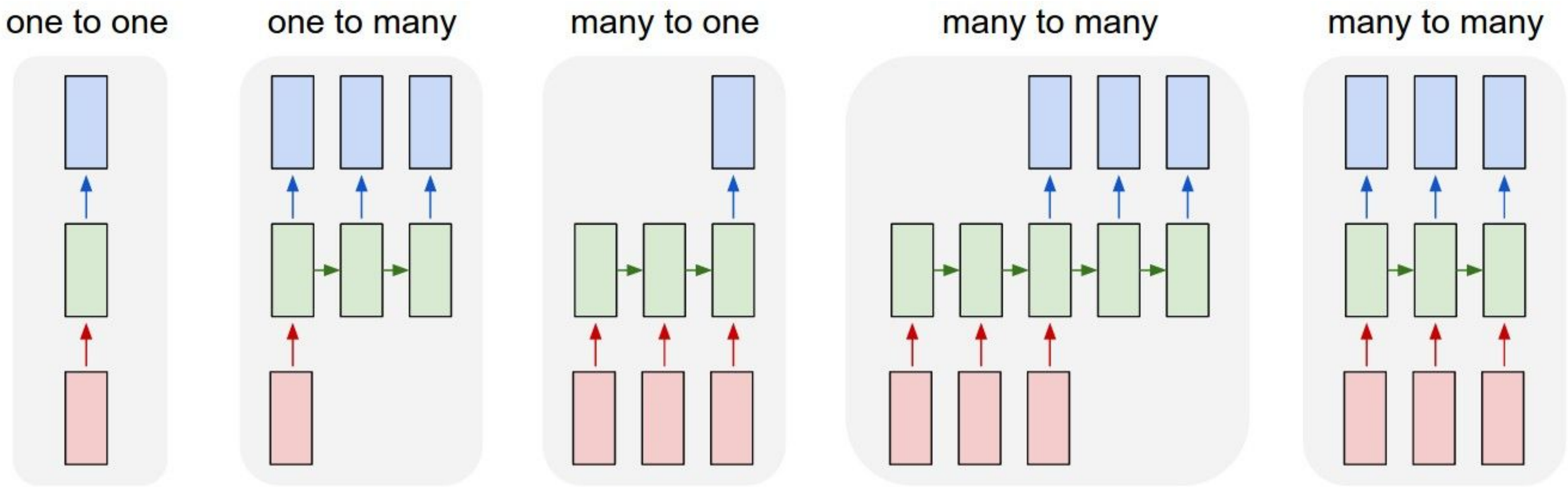
Simple recurrent NNs (SRNs):

- Were difficult to train.
- Had problems when modeling long time dependencies.

Core idea:

- Maintain state.
  - Note: all models considered so far in this course were stateless (timeless).
- ‘Self-control’ by gating signals.

Usage scenarios



‘Vanilla’ LSTM

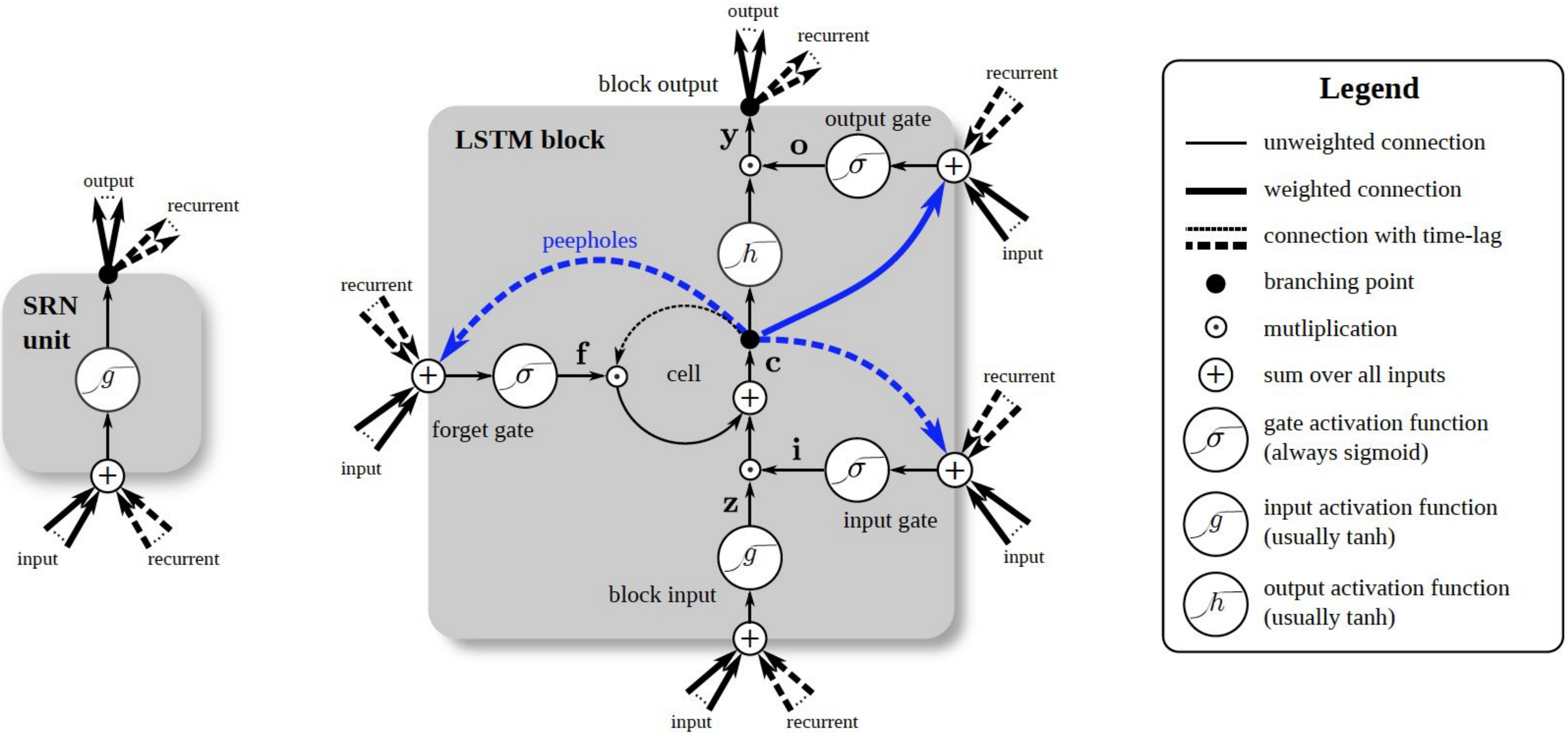
Slightly upgraded compared to the 1997 version.

Core components:

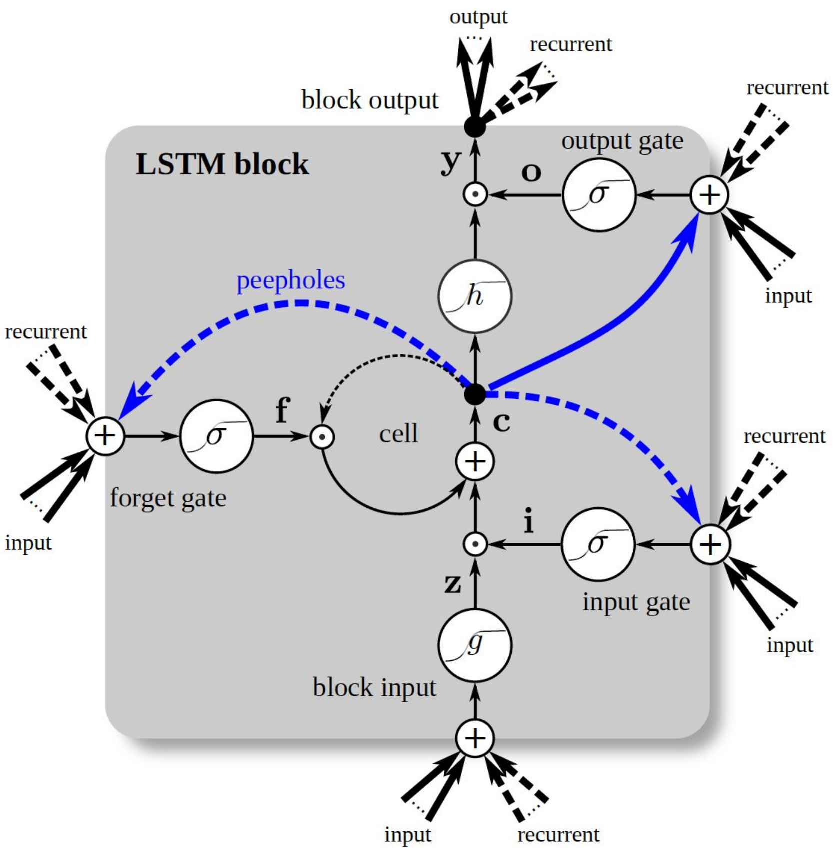
- Three gates:
  - input gate: decides how much the current input should influence the state,
  - forget gate: can reset the state of the cell,
  - output gate: decides how much of the current state should be passed to the next layer.
- Block input
- Single cell (Constant Error Carousel)
- Output activation function
- Peephole connections [optional]

The output goes back to input and all the gates.

Vanilla LSTM



Details on signal propagation



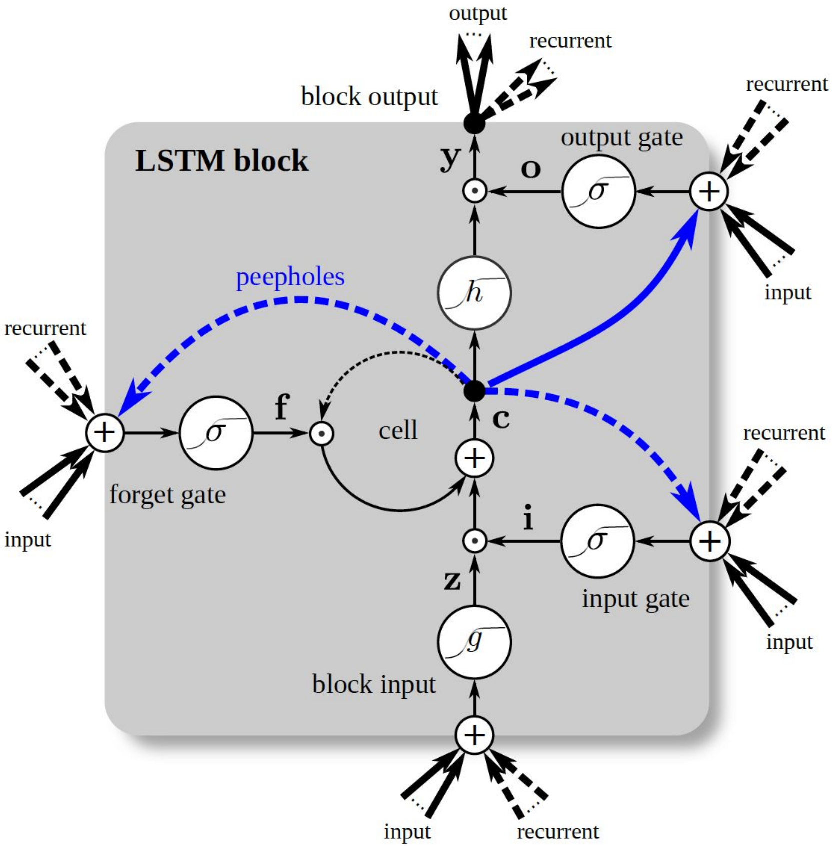
Input x^t: a vector of length M

$$
\begin{aligned}
\bar{\mathbf{z}}^t &= \mathbf{W}_z\mathbf{x}^t + \mathbf{R}_z\mathbf{y}^{t-1} + \mathbf{b}_z && \\
\mathbf{z}^t &= g(\bar{\mathbf{z}}^t) && \textit{block input} \\
\bar{\mathbf{i}}^t &= \mathbf{W}_i\mathbf{x}^t + \mathbf{R}_i\mathbf{y}^{t-1} + \mathbf{p}_i \odot \mathbf{c}^{t-1} + \mathbf{b}_i && \\
\mathbf{i}^t &= \sigma(\bar{\mathbf{i}}^t) && \textit{input gate} \\
\bar{\mathbf{f}}^t &= \mathbf{W}_f\mathbf{x}^t + \mathbf{R}_f\mathbf{y}^{t-1} + \mathbf{p}_f \odot \mathbf{c}^{t-1} + \mathbf{b}_f && \\
\mathbf{f}^t &= \sigma(\bar{\mathbf{f}}^t) && \textit{forget gate} \\
\mathbf{c}^t &= \mathbf{z}^t \odot \mathbf{i}^t + \mathbf{c}^{t-1} \odot \mathbf{f}^t && \textit{cell} \\
\bar{\mathbf{o}}^t &= \mathbf{W}_o\mathbf{x}^t + \mathbf{R}_o\mathbf{y}^{t-1} + \mathbf{p}_o \odot \mathbf{c}^t + \mathbf{b}_o && \\
\mathbf{o}^t &= \sigma(\bar{\mathbf{o}}^t) && \textit{output gate} \\
\mathbf{y}^t &= h(\mathbf{c}^t) \odot \mathbf{o}^t && \textit{block output}
\end{aligned}
$$

Parameters

- M: number of inputs (input ‘channels’)
- N: number of LSTM blocks (in a layer),
  - i.e. the number of outputs from the layer
- Weight matrices:
  - Input weights: $\mathbf{W}_z, \mathbf{W}_i, \mathbf{W}_f, \mathbf{W}_o \in \mathbb{R}^{N \times M}$
  - Recurrent weights: $\mathbf{R}_z, \mathbf{R}_i, \mathbf{R}_f, \mathbf{R}_o \in \mathbb{R}^{N \times N}$
  - Peephole weights: $\mathbf{p}_i, \mathbf{p}_f, \mathbf{p}_o \in \mathbb{R}^{N}$
  - Bias weights: $\mathbf{b}_z, \mathbf{b}_i, \mathbf{b}_f, \mathbf{b}_o \in \mathbb{R}^{N}$
- Implications:
  - All internal quantities and the output are vectors of length N
  - For a single LSTM block (N=1), internal quantities are scalars
  - ‘Mixing’ of inputs (M -> N dimensions) occurs only at the entries to the cell.
  - For a sequence of T elements of dimensionality M (TxM tensor), a LSTM cell produces a sequence of T elements of dimensionality N (TxN tensor)
  - Independently of that, batching is typically possible too (e.g. in TensorFlow)

Training: Backpropagation through time



Δ^t: the vector of deltas passed down from the layer above

$$
\begin{aligned}
\delta \mathbf{y}^t &= \Delta^t + \mathbf{R}_z^T \delta \mathbf{z}^{t+1} + \mathbf{R}_i^T \delta \mathbf{i}^{t+1} + \mathbf{R}_f^T \delta \mathbf{f}^{t+1} + \mathbf{R}_o^T \delta \mathbf{o}^{t+1} \\
\delta \bar{\mathbf{o}}^t &= \delta \mathbf{y}^t \odot h(\mathbf{c}^t) \odot \sigma'(\bar{\mathbf{o}}^t) \\
\delta \mathbf{c}^t &= \delta \mathbf{y}^t \odot \mathbf{o}^t \odot h'(\mathbf{c}^t) + \mathbf{p}_o \odot \delta \bar{\mathbf{o}}^t + \mathbf{p}_i \odot \delta \bar{\mathbf{i}}^{t+1} \\
&\quad + \mathbf{p}_f \odot \delta \bar{\mathbf{f}}^{t+1} + \delta \mathbf{c}^{t+1} \odot \mathbf{f}^{t+1} \\
\delta \bar{\mathbf{f}}^t &= \delta \mathbf{c}^t \odot \mathbf{c}^{t-1} \odot \sigma'(\bar{\mathbf{f}}^t) \\
\delta \bar{\mathbf{i}}^t &= \delta \mathbf{c}^t \odot \mathbf{z}^t \odot \sigma'(\bar{\mathbf{i}}^t) \\
\delta \bar{\mathbf{z}}^t &= \delta \mathbf{c}^t \odot \mathbf{i}^t \odot g'(\bar{\mathbf{z}}^t)
\end{aligned}
$$

Deltas for the inputs (to be backpropagated to the preceding layer):

$$\delta \mathbf{x}^t = \mathbf{W}_z^T \delta \bar{\mathbf{z}}^t + \mathbf{W}_i^T \delta \bar{\mathbf{i}}^t + \mathbf{W}_f^T \delta \bar{\mathbf{f}}^t + \mathbf{W}_o^T \delta \bar{\mathbf{o}}^t$$

Training: Backpropagation through time

$$
\begin{aligned}
\bar{\mathbf{z}}^t &= \mathbf{W}_z \mathbf{x}^t + \mathbf{R}_z \mathbf{y}^{t-1} + \mathbf{b}_z \\
\mathbf{z}^t &= g(\bar{\mathbf{z}}^t) && \textit{block input} \\
\bar{\mathbf{i}}^t &= \mathbf{W}_i \mathbf{x}^t + \mathbf{R}_i \mathbf{y}^{t-1} + \mathbf{p}_i \odot \mathbf{c}^{t-1} + \mathbf{b}_i \\
\mathbf{i}^t &= \sigma(\bar{\mathbf{i}}^t) && \textit{input gate} \\
\bar{\mathbf{f}}^t &= \mathbf{W}_f \mathbf{x}^t + \mathbf{R}_f \mathbf{y}^{t-1} + \mathbf{p}_f \odot \mathbf{c}^{t-1} + \mathbf{b}_f \\
\mathbf{f}^t &= \sigma(\bar{\mathbf{f}}^t) && \textit{forget gate} \\
\mathbf{c}^t &= \mathbf{z}^t \odot \mathbf{i}^t + \mathbf{c}^{t-1} \odot \mathbf{f}^t && \textit{cell} \\
\bar{\mathbf{o}}^t &= \mathbf{W}_o \mathbf{x}^t + \mathbf{R}_o \mathbf{y}^{t-1} + \mathbf{p}_o \odot \mathbf{c}^t + \mathbf{b}_o \\
\mathbf{o}^t &= \sigma(\bar{\mathbf{o}}^t) && \textit{output gate} \\
\mathbf{y}^t &= h(\mathbf{c}^t) \odot \mathbf{o}^t && \textit{block output}
\end{aligned}
$$

Δ^t: the vector of deltas passed down from the layer above

$$
\begin{aligned}
\delta \mathbf{y}^t &= \Delta^t + \mathbf{R}_z^T \delta \mathbf{z}^{t+1} + \mathbf{R}_i^T \delta \mathbf{i}^{t+1} + \mathbf{R}_f^T \delta \mathbf{f}^{t+1} + \mathbf{R}_o^T \delta \mathbf{o}^{t+1} \\
\delta \bar{\mathbf{o}}^t &= \delta \mathbf{y}^t \odot h(\mathbf{c}^t) \odot \sigma'(\bar{\mathbf{o}}^t) \\
\delta \mathbf{c}^t &= \delta \mathbf{y}^t \odot \mathbf{o}^t \odot h'(\mathbf{c}^t) + \mathbf{p}_o \odot \delta \bar{\mathbf{o}}^t + \mathbf{p}_i \odot \delta \bar{\mathbf{i}}^{t+1} \\
&\quad + \mathbf{p}_f \odot \delta \bar{\mathbf{f}}^{t+1} + \delta \mathbf{c}^{t+1} \odot \mathbf{f}^{t+1} \\
\delta \bar{\mathbf{f}}^t &= \delta \mathbf{c}^t \odot \mathbf{c}^{t-1} \odot \sigma'(\bar{\mathbf{f}}^t) \\
\delta \bar{\mathbf{i}}^t &= \delta \mathbf{c}^t \odot \mathbf{z}^t \odot \sigma'(\bar{\mathbf{i}}^t) \\
\delta \bar{\mathbf{z}}^t &= \delta \mathbf{c}^t \odot \mathbf{i}^t \odot g'(\bar{\mathbf{z}}^t)
\end{aligned}
$$

Deltas for the inputs (to be backpropagated to the preceding layer):

$$
\delta \mathbf{x}^t = \mathbf{W}_z^T \delta \bar{\mathbf{z}}^t + \mathbf{W}_i^T \delta \bar{\mathbf{i}}^t + \mathbf{W}_f^T \delta \bar{\mathbf{f}}^t + \mathbf{W}_o^T \delta \bar{\mathbf{o}}^t
$$

Training: Backpropagation through time

Final step: gradients aggregated over the considered time horizon [0,T]

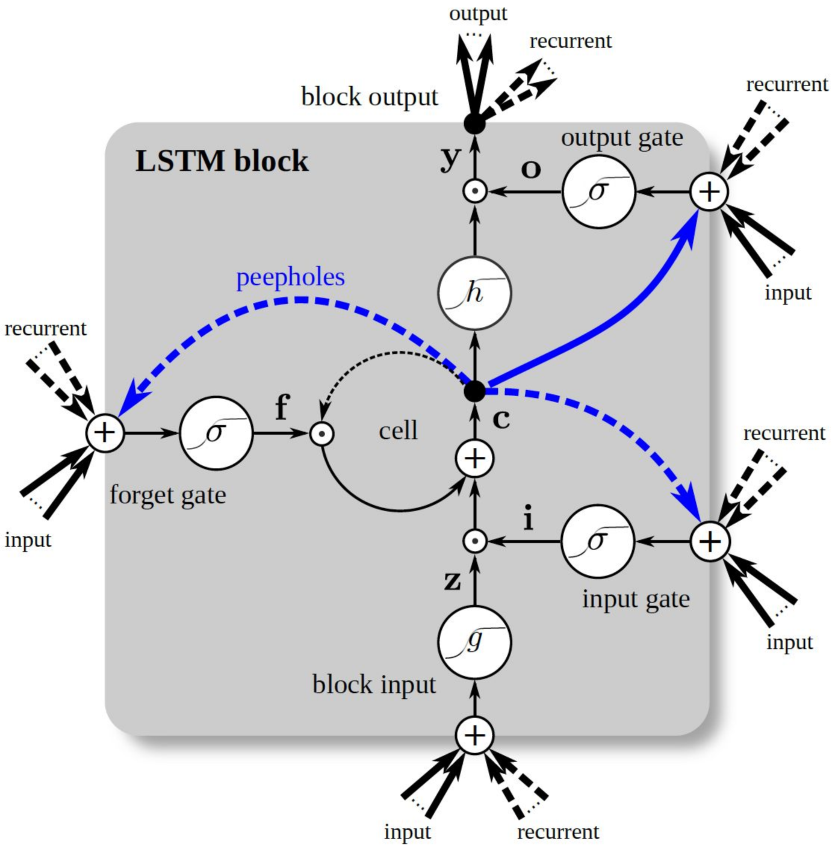
$$\delta \mathbf{W}_\star = \sum_{t=0}^{T} \langle \delta \star^t, \mathbf{x}^t \rangle \qquad \delta \mathbf{p}_i = \sum_{t=0}^{T-1} \mathbf{c}^t \odot \delta \bar{\mathbf{i}}^{t+1}$$

$$\delta \mathbf{R}_\star = \sum_{t=0}^{T-1} \langle \delta \star^{t+1}, \mathbf{y}^t \rangle \qquad \delta \mathbf{p}_f = \sum_{t=0}^{T-1} \mathbf{c}^t \odot \delta \bar{\mathbf{f}}^{t+1}$$

$$\delta \mathbf{b}_\star = \sum_{t=0}^{T} \delta \star^t \qquad \delta \mathbf{p}_o = \sum_{t=0}^{T} \mathbf{c}^t \odot \delta \bar{\mathbf{o}}^t$$

where star denotes any of the internal quantities (before being passed through nonlinearity, i.e. those with dashes); ⟨⟩is scalar product.

Observations



LSTM features two levels of recursion:

- internal, aimed at maintaining the state c,
- external, resulting from y being fed back to the cell in each iteration.

Peephole connections: used only in most sophisticated variants, meant to allow the state to control the gates ‘immediately’.

- Without them, there’s always a lag of 1.

LSTM variants

“Ablated” variants:

- NIG: No Input Gate: $i^t = 1$
- NFG: No Forget Gate: $f^t = 1$
- NOG: No Output Gate: $o^t = 1$
- NIAF: No Input Activation Function: $g(x) = x$
- NOAF: No Output Activation Function: $h(x) = x$
- CIFG: Coupled Input and Forget Gate: $f^t = 1 - i^t$

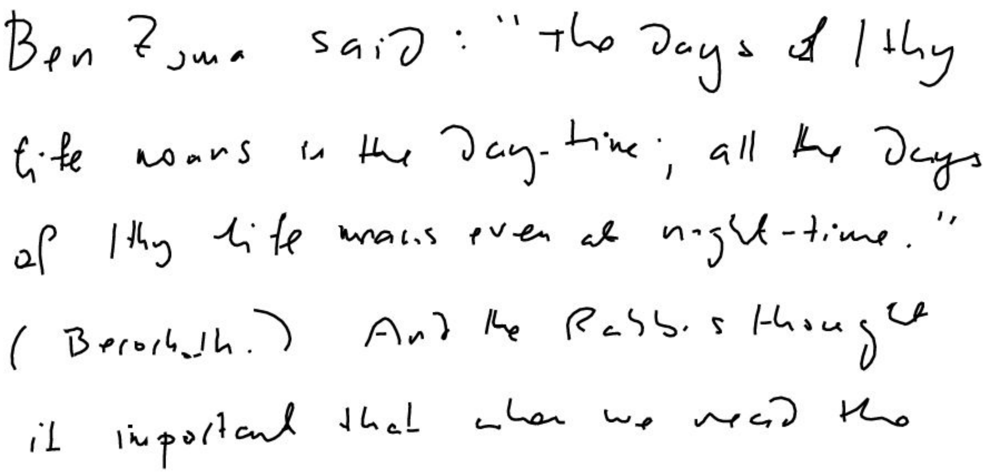
More complex:

- NP: No Peepholes:
- FGR: Full Gate Recurrence:

Demos shown in the paper (‘Space odyssey’)

- TIMIT: speech recognition
  - Input: 12 MFCCs (Mel Frequency Cepstrum Coefficients) + energy
  - Task: classification of phonemes
- IAM Online: The IAM Online Handwriting Database
  - Inputs: pen positions
  - Outputs: characters
  - Task: mapping
- JSB Chorales:
  - Inputs: MIDI sequences of 382 JS Bach chorales transposed to C major or C minor, sampled every quarter note
  - Task: next-step prediction
  - Loss function: minimizing neg log-likelihood
  - A sample (other authors): https://www.youtube.com/watch?v=Iz8xQou2OqA

IAM Online example



(a)

```
Ben Zoma said: "The days of 1thy
life means in the day-time; all the days
of 1thy life means even at night-time ."
(Berochoth .) And the Rabbis thought
it important that when we read the
```

(b)

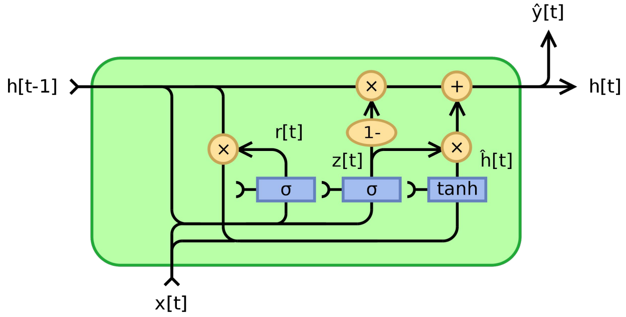
Notable variants (by other authors)

Gated Recurrent Unit (GRU)

Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, Yoshua Bengio

https://arxiv.org/abs/1412.3555



Key differences w.r.t. canonical LSTM:

- no peephole connections,
- no output activation function,
- the input and forget gate coupled into an update gate.

GRUs vs. LSTM

- GRU has fewer parameters than LSTM.
  - Easier training.
- GRU's performance on certain tasks of polyphonic music modeling and speech signal modeling was found to be similar to that of LSTM.
- GRUs have been shown to exhibit even better performance on certain smaller datasets.

However,

- LSTM is "strictly stronger" than the GRU as it can easily perform unbounded counting, while the GRU cannot
  - Weiss, Goldberg, Yahav, *On the Practical Computational Power of Finite Precision RNNs for Language Recognition*, 2018.
- That's why GRU tnds to fail to learn in certain domains that are learnable by the LSTM.

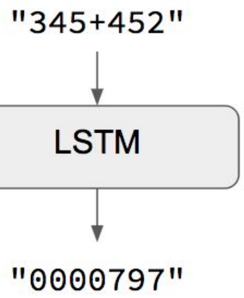
Other observations concerning RNNs

- Can be applied to any sequences, not only time sequences
- Can be used in bi-directional mode
  - Technically: a pair of LSTM cells: forward, backward.
- Fare pretty well on sequences that are meant to represent nonlinear structures, e.g. trees
  - Example: Trees in prefix notation: (+ (* 2 7) (- 3 x))
- Paved the way for recurrent architectures capable of processing non-linear data structures, e.g. trees.

Notable examples of RNN usage

The Seq2seq blueprint

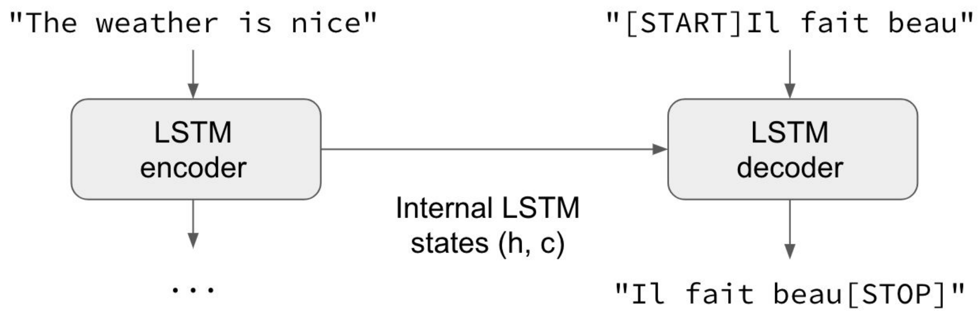
- A broad class of tasks that consist in mapping sequences to sequences.
- The simplest variant: mapping of sequences of same length.
- Can be implemented with a single RNN, e.g. LSTM or GRU.
- Example: training an RNN to add numbers encoded as character strings:



- Notice the padding zeros.
- See: *A ten-minute introduction to sequence-to-sequence learning in Keras*, F. Chollet, https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html

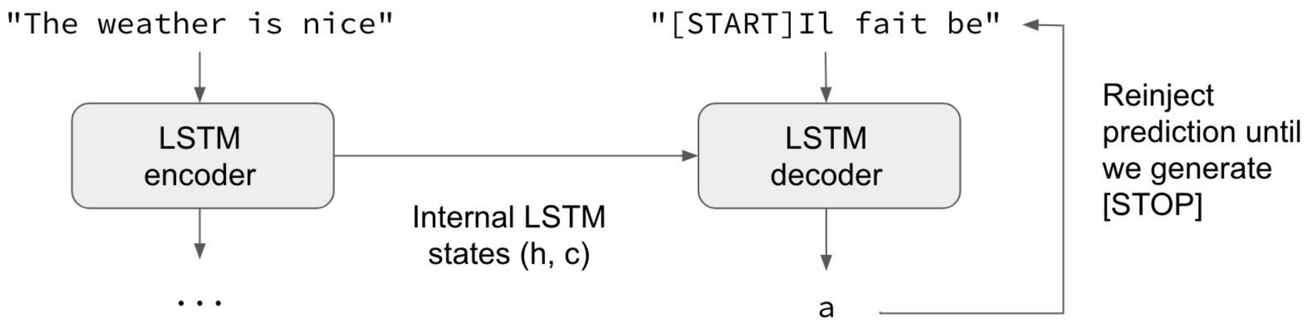
The Seq2seq blueprint

- The case for non-equal lengths: “canonical” seq2seq\
- Two RNNs (e.g. LSTMs): encoder and decoder
- Encoder LSTM “folds” the input sequence into its hidden state h (i.e. a fixed-length latent representation)
- Decoder LSTM’s state is initialized with h and is trained to predict the next token in the target sequence given the previous tokens of the target sequence as input (it should reproduce it’s input shifted by one token) – *teacher forcing*.



Querying of the canonical seq2seq model

- The consecutive tokens produced by the decoder are fed back to its input.



- Notice: this scheme can be also used to train the decoder without teacher forcing.

The Seq2seq blueprint

- Used widely in Machine Translation, Text Summarization, Conversational Modeling, and more
- Typically accompanied with word embeddings.
  - Embedding = a mapping from categorical domain to Cartesian space.
  - The basic form of embedding: learnable look-up table of n entries (dictionary size), each containing an (initially random) vector of m reals (embedding dimensionality).
  - Notable representatives: Glove, Word2Vec, FastText, ELMo, ...
- A vast range of variants
  - Often involves additional over-the-sequence attention mechanisms
- Perform very well
  - One of the cornerstones of Neural Machine Translation (NMT)

More on similar models: Linguistic Engineering (Inżynieria Lingwistyczna), mgr Mateusz Lango

Recommended reading

Andrej Karpathy blog
The Unreasonable Effectiveness of Recurrent Neural Networks,
May 21, 2015
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

```
PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

Second Lord:
They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:
Come, sir, I will make did behold your worship.

VIOLA:
I'll drink it.
```

An example of follow-up of the Seq2Seq paradigm

Tree2tree (recurrent) autoencoder

Ain't Nobody Got Time for Coding: Structure-Aware Program Synthesis from Natural Language

Jakub Bednarek, Karol Piaskowski, Krzysztof Krawiec

https://arxiv.org/abs/1810.09717

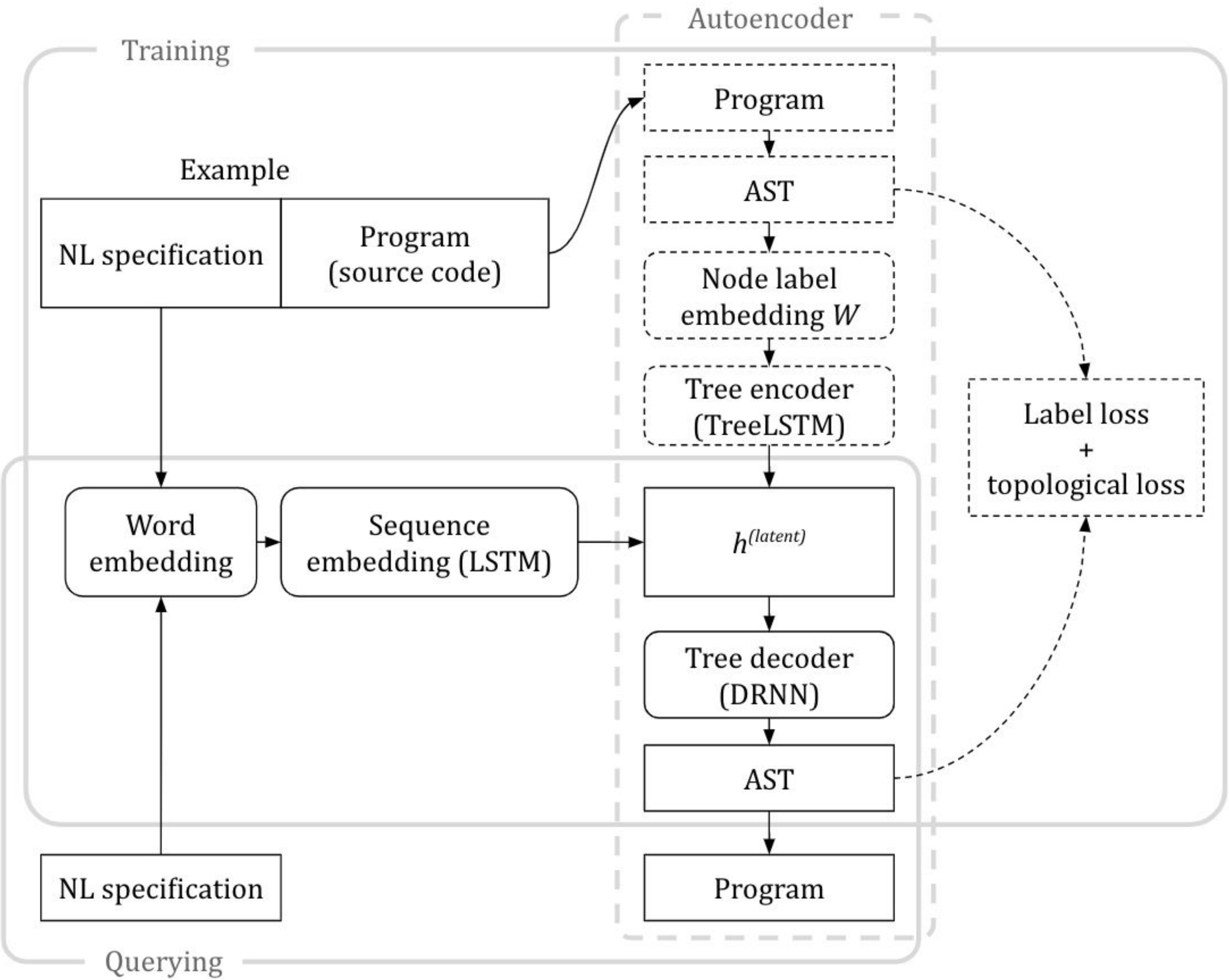
Motivations

Program synthesis: producing programs from specifications.

Forms of specifications used traditionally in PS:

- examples,
- formal specifications,
- partial programs.

What is the most natural form of specification from the human viewpoint?


Autoencoder
Training
Program
Example
AST
NL specification
Program
(source code)
Node label
embedding W
Tree encoder
(TreeLSTM)
Label loss
+
topological loss
Word
embedding
Sequence
embedding (LSTM)
$h^{(latent)}$
Tree decoder
(DRNN)
AST
NL specification
Program
Querying

The benchmark (Polosukhin & Skidanov, 2018)

- Functional domain-specific language based on Lisp (AlgoLisp)
- Three types: string, bool, function
- The grammar:

```
program       ::= symbol
symbol        ::= constant | argument | function_call | function | lambda
constant      ::= number | string | True | False
function_call ::= (function_name arguments)
function      ::= function_name
arguments     ::= symbol | arguments , symbol
function_name ::= Reduce | Filter | Hap | Head | + | - | ...
lambda        ::= lambda function_call
```

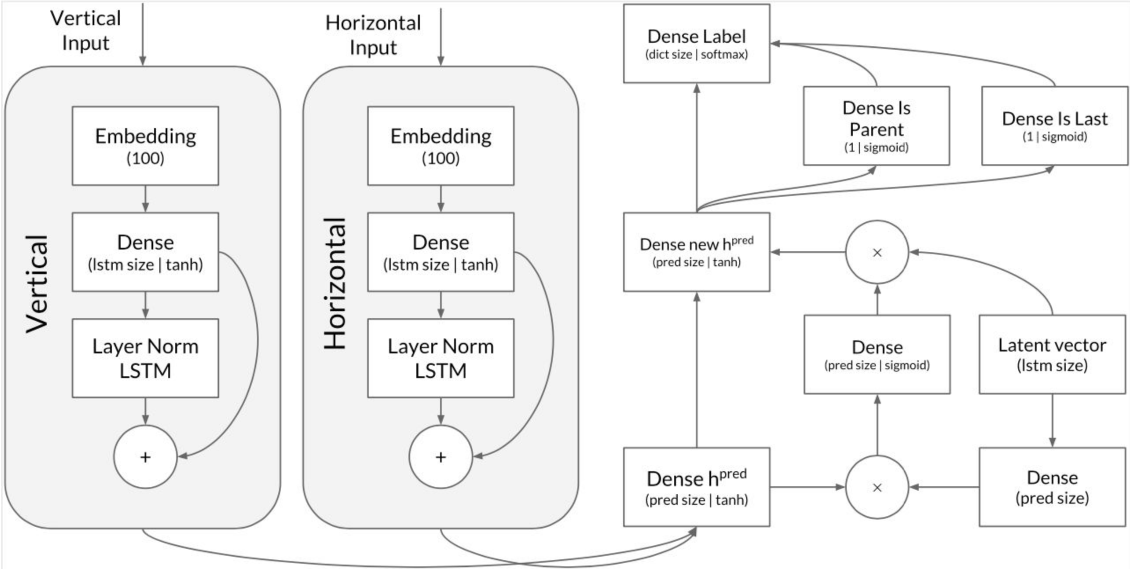
- An example of an (input, output) pair:

You are given an array a. Find the smallest element in a, which is strictly greater than the minimum element in a
`(reduce (filter a (partial0 (reduce a inf) <)) inf min)`

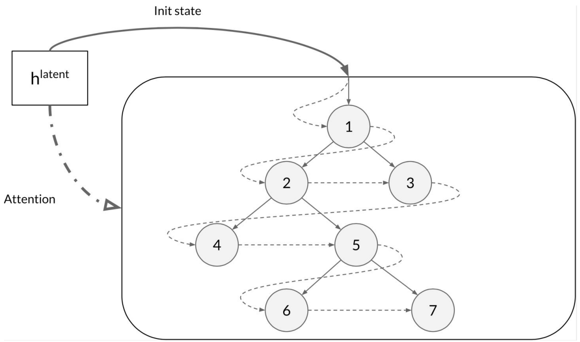
- 99506 examples in total, split into training (79214) validation (9352) and test set (10940 examples).

The decoder

- Doubly-recurrent NN



- Traversal of the output tree



Some results

Percentage of perfectly synthesized programs (i.e., syntactically identical to the target ones) for SAPS configurations trained from scratch:

Model	Validation	Test	Number of parameters [M]
SAPS V	0%	0%	5.06
SAPS V Att	0%	0%	5.17
SAPS H	84.31%	78.92%	5.06
SAPS H Att	89.54%	86.20%	5.17
SAPS VH	93.65%	89.10%	5.62
SAPS VH Att	**93.73%**	**92.36%**	5.73

Comparison with other models:

Model	Validation	Test
Attentional Seq2Seq	54.4%	54.1%
Seq2Tree	61.2%	61.0%
SAPSpre VH Att (256)	**86.67%**	**83.80%**
Seq2Tree + Search	86.1%	85.8%

Notice the relatively good performance of Seq2Seq!

Examples of synthesized programs

Specification	Synthesized program
you are given numbers a and b, your task is to find a + b	`(+, a, b)`
you given numbers a b, your is find a + b	`(+, a, b)`
*given **a numbers b**, find a + b*	`(+, a, b)`
given a numbers b, a + b	`(+, a, b)`
a b, a + b	`(+, (+, a, b), c)`
*you are given numbers a and b, your task is to find **a multiplied by b***	`(*, a, b)`
*you are given numbers a and b, your task is to find **minimum a and b***	`(min, a, b)`
*given a number a and an array of numbers b, find the length of the longest subsequence of range from **0 to a inclusive** that is a prefix of b*	`(reduce, (range, 0, (+, a, 1)), 0, (lambda2, (if, (==, arg2, (if, (<, arg1, (len, b)), (deref, b, arg1), 0)), (+, arg1, 1), arg1)))`
*given a number a and an array of numbers b, find the length of the longest subsequence of range from **1 to a exclusive** that is a prefix of b*	`(reduce, (range, 1, a), 0, (lambda2, (if, (==, arg2, (if, (<, arg1, (len, b)), (deref, b, arg1), 0)), (+, arg1, 1), arg1)))`
given an array of numbers a, find median of values in a after only keeping first half	`(deref, (sort, (slice, a, 0, (/, (len, a), 2))), (/, (len, (slice, a, 0, (/, (len, a), 2))), 2))`
given an array of numbers a, find mean of values in a after only keeping second half	`(/, (reduce, (slice, a, (/, (len, a), 2), (len, a)), 0, +), (len, (slice, a, (/, (len, a), 2), (len, a))))`