Beyond RNN

Multi-dimensional RNN, RNN grammars, transducers and Turing machines

Dávid Márk Nemeskey

MTA SZTAKI

April 27, 2017

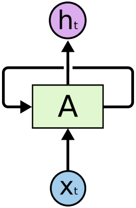
Overview

1. A short reminder
2. Multidimensionality
3. "Chomsky" networks
   - Turing machines
   - Transducers
   - (P)CFG parsers

Recurrent Neural Networks

What is an RNN?

- A recursive network structure
- Especially suited for sequence modeling
- Trained with Backpropagation Through Time (BPTT)



[1]

[1] Images stolen from http://colah.github.io/posts/2015-08-Understanding-LSTMs/

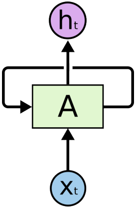
Recurrent Neural Networks

What is an RNN?

- A recursive network structure
- Especially suited for sequence modeling
- Trained with Backpropagation Through Time (BPTT)

It has problems though...

- BPTT theoretically must recurse back to $t = 0$



[1]

[1] Images stolen from http://colah.github.io/posts/2015-08-Understanding-LSTMs/

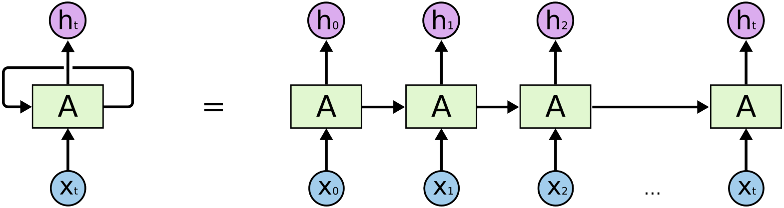
Recurrent Neural Networks

What is an RNN?

- A recursive network structure
- Especially suited for sequence modeling
- Trained with Backpropagation Through Time (BPTT)

It has problems though...

- BPTT theoretically must recurse back to $t = 0$



[1]

[1] Images stolen from `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

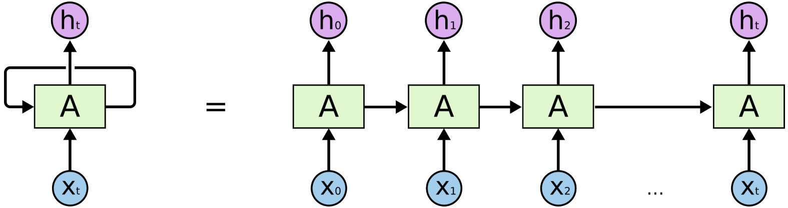
Recurrent Neural Networks

What is an RNN?

- A recursive network structure
- Especially suited for sequence modeling
- Trained with Backpropagation Through Time (BPTT)

It has problems though...

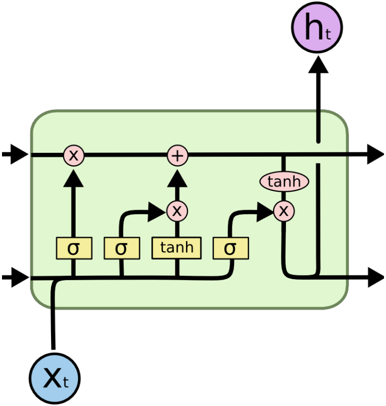
- BPTT theoretically must recurse back to $t = 0$
- Exploding and vanishing gradients



[1]

[1] Images stolen from `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

Long Short-Term Memory (LSTM)

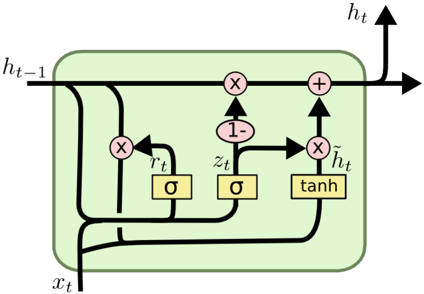


$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f])$$
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i])$$
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o])$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$
$$h_t = o_t * \tanh(C_t)$$

- Introduced in Hochreiter and Schmidhuber, (1997)
- Forget gate added in Gers et al., (2000)
- Double state: C and h

Gated Recurrent Unit (GRU)

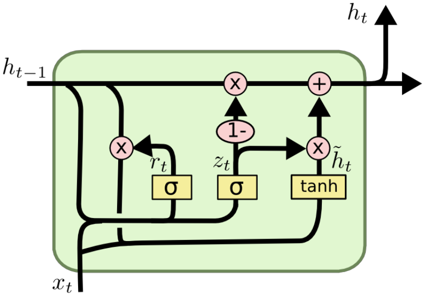


$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z])$$
$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r])$$
$$\tilde{h}_t = \tanh(W_h \cdot [r_t * h_{t-1}, x_t] + b_h)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

- Introduced in Cho et al., (2014)
- A bit closer to regular RNN (single state)
- LSTM seems to be better for language modeling (Jozefowicz et al., 2015)

Gated Recurrent Unit (GRU)



$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z])$$
$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r])$$
$$\tilde{h}_t = \tanh(W_h \cdot [r_t * h_{t-1}, x_t] + b_h)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

- Introduced in Cho et al., (2014)
- A bit closer to regular RNN (single state)
- LSTM seems to be better for language modeling (Jozefowicz et al., 2015)
- Not a planar graph!!!

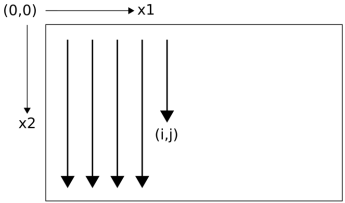
Multi-Dimensional RNN[2]

- Multi-dimensional data: images, videos, fMRI, etc.
- RNNs are tailored to sequential, not multi-dimensional, data
- CNNs: most successful, but
  - hand specified kernel sizes
  - don't scale well to large images (???)
- HMMs: multi-dimensional variants exist, but
  - Viterbi's time complexity is exponential in the number of data points
  - the number of transition probabilities is exponential in the number of dimensions
- RNNs: the data must be linearized
- MDRNNs do not suffer from these problems

[2] Graves et al., (2007)

Multi-Dimensional RNN (cont.)

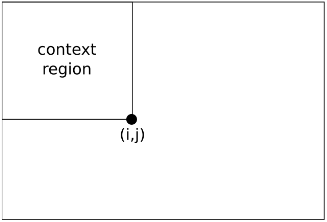
- Same idea as in 1D: the RNNs runs on a data sequence
- However, when processing x_{ijk} to compute h_{ijk}, the MDRNN receives all previous neighboring activations: h_{i-1jk}, h_{ij-1k}, h_{ijk-1}.
- An ordering must be defined on the data points $(x_1, ..., x_n)$ that ensures that these are available.
- One example: $(x_1, ..., x_n) < (x'_1, ..., x'_n)$ if $\exists m \in 1, .., n$: $x_m < x'_m$ and $x_i = x'_i \forall i \in (1, ..., m-1)$.



- Then, since MDRNN requires a FW and a BW pass per iteration, training is linear in n and $|W|$.

Multi-Dimensional RNN (cont.)

- Multi-directional MDRNNs
  - Generalization of BRNN in 1D
  - Sees the context in all directions
  - 2^n hidden layers
- MDLSTM
  - n forget gates



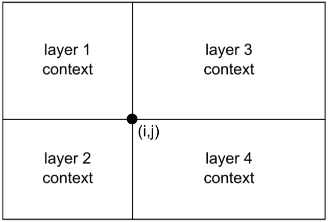


Figure 1: Contexts available to single- and multi-directional 2D RNNs

Multi-Dimensional RNN: experiments

- Segmentation on the Air Freight DB
  - 120×160 images
  - MDLSTM with 2^2 hidden layers, 25 cells each
  - Pixel classification error was 7.3%
- MNIST
  - Compared it with the (then-)best CNN
  - Also tested the models on a “warped” test set (elastic deformations applied on the images)
  - No data augmentation during training

Table 1: Image error rates on MNIST

Algorithm	Clean Test Set	Warped Test Set
MDRNN	1.1%	6.8%
CNN	0.9%	11.3%

Grid Long Short-Term Memory[3]

- Motivation
  - Deep networks (number of layers, **not** unrolling) are key to finding complex patterns ("Stacked LSTM")
  - Same problems with unrolling: vanishing gradient
  - Idea: use the LSTM principle between layers as well
- A generalization of LSTM / MDRNN
- Depth is treated as just another dimension

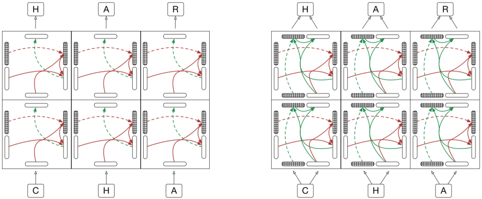


Figure 2: Stacked (left) vs. 2D Grid (right) LSTM

[3]Kalchbrenner et al., (2016)

Grid Long Short-Term Memory (cont.)

- Architecture
  - Same as MDRNN: N inputs per block; different: N outputs
  - In a block, all dimensions are processed in parallel; except for *priority dimensions*
  - Can include non-LSTM dimensions (depth is non-LSTM $\equiv$ MDRNN)
  - Weight sharing: along any dimension. All: *Tied N-LSTM*
- Experiments
  - Addition of 15-digit numbers (2D): 99%
  - Memorization of 20 symbols (2D): 99%
  - Character-level LM (Hutter challenge) (2D): 1.47
  - Translation (3D): 60.2
- Results
  - In all experiments, tied N-LSTM $>$ N-LSTM $>$ LSTM
  - Outperforms previous approaches (of course)

RNNs and state machines

- RNNs can be considered as the distributed version of state machines
- Weighted SM's can be simulated with beam search
- Let's see how the networks corresponding to various levels of the Chomsky hierarchy look like:
  - Turing machines
  - Transducers
  - (P)CFG parsers
- (Note: similarly, the encoder-decoder framework could also be thought of as a variant of reinforcement learning.)

Neural Turing Machines[4]

- Architecture
  - *Controller*: a usual FF or RNN which also accesses the memory via the
  - *Read and write heads*: accesses memory locations
  - *Memory*: N M-long vectors
  - (But: is this really a Turing machine?)
  - Fully differentiable
- Memory access is *soft*:
  - Reads: $r_t \leftarrow \sum_i w_t(i) M_t(i)$
  - Writes (*erase* and *add*): $M_t(i) \leftarrow M_{t-1}[1 - w_t(i)e_t] + w_t(i)a_t$
- Memory addressing:
  - *Content-based addressing*: find locations whose values are similar ($\cos$) to x. Like cache.
  - *Location-based addressing*: like RAM. Random access and iteration via a *shift* mechanism.

[4] Graves et al., (2014)

Neural Turing Machines: experiments

Compare NTM's performance to LSTM

- Sequence copy:
  - trained on 1–20, generalizes well up to 120
  - LSTM degrades quickly above 20
- Repeat copy:
  - trained on 1–10
  - both learn the task. NTM outputs EOS after each copy after 10 :)
- Associative recall: given a list L, return l_{i+1} if l_i is queried
  - much better then LSTM: 0 error in 30k iteration vs. errors after 1M
- Dynamic n-grams: 6-gram over bits
  - neither reach the optimal predictor
- Priority sort
  - NTM seems to have used priorities as relative pointers

NTM outperforms LSTM: explicit memory is better than implicit.

Neural Turing Machines: relate work

- Memory Networks (Weston et al., 2015)
  - Augments a network with a (large) memory
  - Tailored for question answering
  - Not an end-to-end NN; the whole system is a hodge-podge of tricks
  - Good results for QA
- Learning to execute (Zaremba and Sutskever, 2014)
  - Trains an LSTM to read a Python(esque) program snippet and return the value of the printed expression
  - Difficult to evaluate, so two smaller tasks were added: addition and memorization
  - Various curriculum learning strategies were tested; the one that worked best randomly mixed harder samples into the real curriculum

Sequence transduction with RNNs [5]

- Motivation
  - RNNs are natural choice for transductions where the alignment is known in advance (sequence classification, language modeling)
  - Problem if $|\boldsymbol{x}| \neq |\boldsymbol{y}|$
- Alignment
  - Input: $\boldsymbol{x} = (x_1, ..., x_t) \in \mathcal{X}^*$, output: $\boldsymbol{y} = (y_1, ..., y_U) \in \mathcal{Y}^*$
  - Extend the output space to $\bar{\mathcal{Y}} = \mathcal{Y} \cup \varnothing$
  - Then the transducer computes $\Pr(\boldsymbol{a} \in \bar{\mathcal{Y}}^* | \boldsymbol{x})$, where $\boldsymbol{a}$ is the *alignment* between $\boldsymbol{x}$ and $\boldsymbol{y}$
  - Finally, $\Pr(\boldsymbol{y} \in \mathcal{Y}^* | \boldsymbol{x}) = \sum_{\boldsymbol{a} \in \mathcal{B}^{-1}(\boldsymbol{y})} \Pr(\boldsymbol{a}|\boldsymbol{x})$
- Architecture
  - *Transcription network* $\mathcal{F}$: scans the input $\boldsymbol{x}$ and outputs $\boldsymbol{f} = (f_1, ..., f_T) \in \bar{\mathcal{Y}}^*$.
  - *Prediction network* $\mathcal{G}$: scans the output $\boldsymbol{y}$ and outputs $\boldsymbol{g} = (g_0, ..., g_U) \in \bar{\mathcal{Y}}^*$.

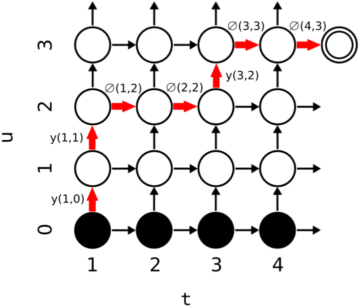
[5] Graves, (2012)

Sequence transduction with RNNs (cont.)

- Alignment lattice of $\boldsymbol{f}$ and $\boldsymbol{g}$
  - Compute the probability of transitions

$$\Pr(k \in \bar{\mathcal{Y}} | t, u) = \frac{\exp(f_t^k + g_u^k)}{\sum_{k' \in \bar{\mathcal{Y}}} \exp(f_t^{k'} + g_t^{k'})}$$

  - $\Pr(\boldsymbol{y}|\boldsymbol{x})$ is the sum across all alignments.



- Forward-backward algorithm
  - Naive calculation is intractable
  - A more effective forward-backword algorithm is defined
  - Further tricks for efficiency
- Testing
  - Fixed-width beam search

Sequence transduction with RNNs: experiments

Table 3: Phoneme recognition results on TIMIT

System	Epochs	Log-loss	Error rate
Prediction	58	4.0	72.9%
CTC	96	1.3	25.5%
Transducer	76	1.0	23.2%
Then-best (DBN + mcRBM)			20.5%

- The lackluster performance might be attributed to the too small dataset
- In a subsequent paper (Graves, 2013), it did not perform well for handwriting synthesis :)

Recurrent Neural Network Grammars [6]

- A transition-based, top-down, discriminative / generative parser.
- Components:
  - Input buffer (for parsing) B: where the sentence sits
  - Output buffer (for generation) T: where the generated words are put
  - Stack S: a Stack LSTM (Dyer et al., 2015)
- Transitions:
  - NT(X): pushes the open nonterminal X (e.g. "(VP") onto the stack
  - SHIFT: removes the terminal x from B and pushes it onto the stack
  - GEN: generates terminal x and puts it on the stack and in T; only for generation.
  - REDUCE: pops the contents of the stack until an open NT is found, then merges these items into a complete constituent and pushes that back on the stack.

[6] Dyer et al., (2016)

Recurrent Neural Network Grammars: an example

Table 5: Generative parsing of the sentence *The hungry cat meows.*

	Stack	Buffer	Action
0		*The* | *hungry* | *cat* | *meows* | .	NT(S)
1	(S	*The* | *hungry* | *cat* | *meows* | .	NT(NP)
2	(S | (NP	*The* | *hungry* | *cat* | *meows* | .	SHIFT
3	(S | (NP | *The*	*hungry* | *cat* | *meows* | .	SHIFT
4	(S | (NP | *The* | *hungry*	*cat* | *meows* | .	SHIFT
5	(S | (NP | *The* | *hungry* | *cat*	*meows* | .	REDUCE
6	(S | (NP *The hungry cat*)	*meows* | .	NT(VP)
7	(S | (NP *The hungry cat*) | (VP	*meows* | .	SHIFT
8	(S | (NP *The hungry cat*) | (VP *meows*	.	REDUCE
9	(S | (NP *The hungry cat*) | (VP *meows*)	.	SHIFT
10	(S | (NP *The hungry cat*) | (VP *meows*) | .	.	REDUCE
11	(S | (NP *The hungry cat*) | (VP *meows*) .)	.	

Recurrent Neural Network Grammars (cont.)

Generative model: learns the joint distribution $p(\boldsymbol{x}, \boldsymbol{y})$.

- State is $\mathbf{u}_t = \tanh(\mathbf{W}[\mathbf{o}_t; \mathbf{s}_t; \mathbf{h}_t] + \mathbf{c})$, computed from the embeddings for T_t, S_t and the action history $\boldsymbol{a}_{\leq t}$.
- The syntactic composition for REDUCE is implemented via a BiLSTM
- Word generation is done in two steps: first predict the action (GEN), then the word with class-factored softmax (Brown clustering).
- Inference via importance sampling, where the proposal distribution is the discriminative model

The discriminative model is similar, it just learns the sequence of actions conditioned on the input.

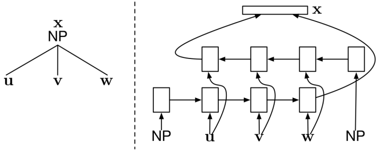


Figure 3: BiLSTM syntactic composition function

Recurrent Neural Network Grammars: experiments

- Experiments:
  - Language modeling and parsing
  - English (PTB) and Chinese (CTB)
  - Best results for parsing (93.3 and 86.9)
  - LM performance is not very good (105.2), but they **LIE** and say it is better than an LSTM LM
- Related work:
  - Grammar as a Foreign Language (Vinyals et al., 2015)
    - Parsing as translation (encoder-decoder and attention)
  - Transition-Based Dependency Parsing with Stack Long Short-Term Memory (Dyer et al., 2015)
    - A discriminative dependency parser with similar architecture
  - Neural Architectures for Named Entity Recognition (Lample et al., 2016)
    - A transition-based NER with Stack LSTM

Thank you for your **attention**

- Maintains a stack pointer (TOP)
- push:
  - append the new element on the right
  - take not $\mathbf{c}_{t-1}$ and $\mathbf{h}_{t-1}$ as input but $\mathbf{c}_{\mathsf{TOP}}$ and $\mathbf{h}_{\mathsf{TOP}}$
- pop just updates the stack pointer

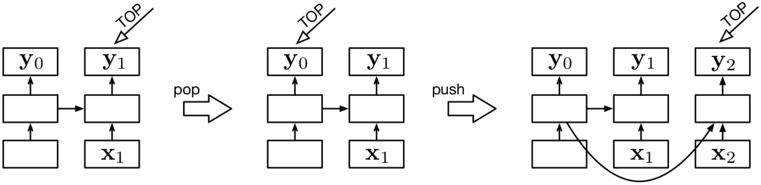


Figure 4: Stack LSTM with a pop and a push operation applied to it

[7] Dyer et al., (2015)

Appendix: bibliography I

Cho, Kyunghyun, Bart van Merriëenboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio (2014). "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar, pp. 1724–1734. URL: http://www.aclweb.org/anthology/D14-1179.

Dyer, Chris, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith (2015). "Transition-Based Dependency Parsing with Stack Long Short-Term Memory". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Beijing, China: Association for Computational Linguistics, pp. 334–343. arXiv: 1505.08075 [cs.CL]. URL: http://www.aclweb.org/anthology/P15-1033.

Dyer, Chris, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith (2016). "Recurrent Neural Network Grammars". In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. San Diego, California: Association for Computational Linguistics, pp. 199–209. arXiv: 1602.07776 [cs.CL]. URL: http://www.aclweb.org/anthology/N16-1024.

Appendix: bibliography II

Gers, Felix A, Jürgen Schmidhuber, and Fred Cummins (2000). "Learning to forget: Continual prediction with LSTM". In: *Neural computation* 12.10, pp. 2451–2471.

Graves, Alex (2012). "Sequence transduction with recurrent neural networks". In: *Representation Learning Workshop, ICML 2012*. Edinburgh, Scotland. arXiv: 1211.3711 [cs.NE].

Graves, Alex (2013). *Generating Sequences With Recurrent Neural Networks*. arXiv: 1308.0850 [cs.NE].

Graves, Alex, Santiago Fernández, and Jürgen Schmidhuber (2007). "Multi-Dimensional Recurrent Neural Networks". In: *Proceedings of the 2007 International Conference on Artificial Neural Networks (ICANN)*. Porto, Portugal. arXiv: 0705.2011 [cs.CV].

Graves, Alex, Greg Wayne, and Ivo Danihelka (2014). *Neural turing machines*. Tech. rep. arXiv: 1410.5401 [cs.NE].

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long Short-Term Memory". In: *Neural Computation* 9.8, pp. 1735–1780.

Jozefowicz, Rafal, Wojciech Zaremba, and Ilya Sutskever (2015). "An empirical exploration of recurrent network architectures". In: *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pp. 2342–2350.

Appendix: bibliography III

Kalchbrenner, Nal, Ivo Danihelka, and Alex Graves (2016). "Grid long short-term memory". In: *Proceedings of the International Conference on Learning Representations*. San Juan, Puerto Rico. arXiv: 1507.01526 [cs.NE].

Lample, Guillaume, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer (2016). "Neural Architectures for Named Entity Recognition". In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. San Diego, California: Association for Computational Linguistics, pp. 260–270. arXiv: 1603.01360 [cs.CL]. URL: http://www.aclweb.org/anthology/N16-1030.

Vinyals, Oriol, Łukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton (2015). "Grammar as a foreign language". In: *Advances in Neural Information Processing Systems*, pp. 2773–2781. arXiv: 1412.7449 [cs.CL]. URL: http://papers.nips.cc/paper/5635-grammar-as-a-foreign-language.pdf.

Weston, Jason, Sumit Chopra, and Antoine Bordes (2015). "Memory networks". In: *International Conference on Learning Representations (ICLR 2015)*. arXiv: 1410.3916 [cs.CL].

Zaremba, Wojciech and Ilya Sutskever (2014). "Learning to execute".