IN5550: Neural Methods in Natural Language Processing
Sub-lecture 6.4
Gated RNNs: LSTM and GRU

Andrey Kutuzov

University of Oslo

24 February 2022

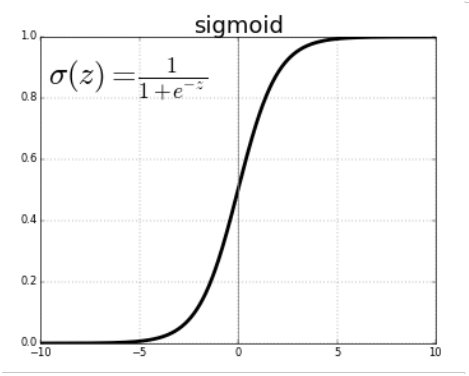


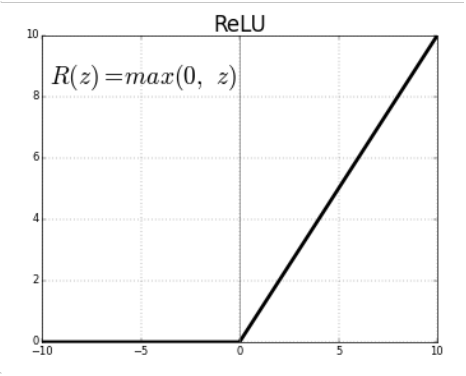
Contents



1. Gated architectures

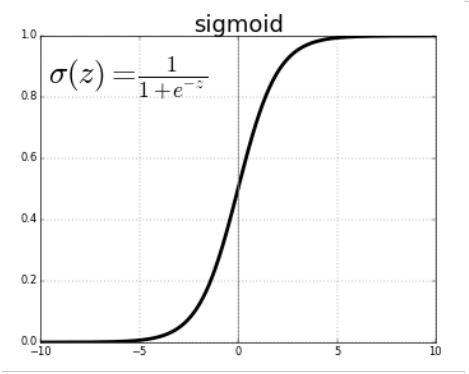
Vanishing (or Exploding) Gradients

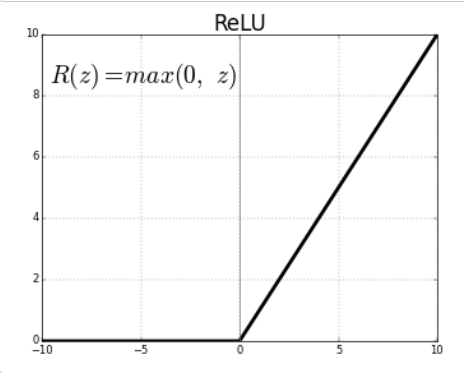




- Shared RNN parameter values can decrease (or grow) monotonically
- gradients for parameters 'deep' down diminish (or get out of hand, 'explode')

Vanishing (or Exploding) Gradients



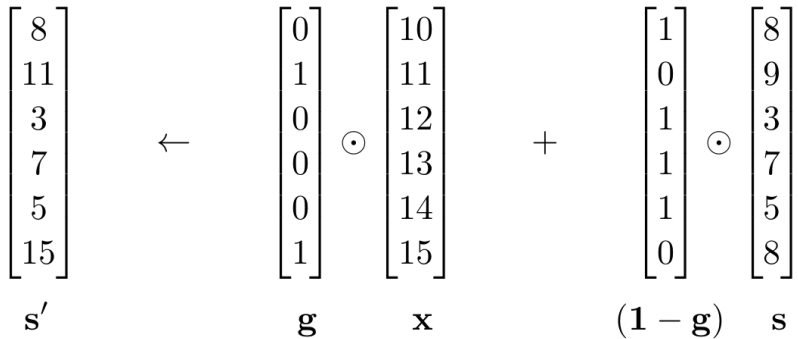


- Shared RNN parameter values can decrease (or grow) monotonically
- gradients for parameters 'deep' down diminish (or get out of hand, 'explode')

→ ineffective back-propagation of error signals through the network

→ can make it difficult for the RNN to learn long-range dependencies

Gatekeeping: Let's filter and control memory access



- Hadamard product ($\odot$) simply performs element-wise multiplication
- vector $\boldsymbol{g}$ acts as a gate: masks out parts of the memory and input

Gatekeeping: Let's filter and control memory access

$$\underbrace{\begin{bmatrix} 8 \\ 11 \\ 3 \\ 7 \\ 5 \\ 15 \end{bmatrix}}_{\mathbf{s}'} \leftarrow \underbrace{\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}}_{\mathbf{g}} \odot \underbrace{\begin{bmatrix} 10 \\ 11 \\ 12 \\ 13 \\ 14 \\ 15 \end{bmatrix}}_{\mathbf{x}} + \underbrace{\begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \end{bmatrix}}_{(\mathbf{1}-\mathbf{g})} \odot \underbrace{\begin{bmatrix} 8 \\ 9 \\ 3 \\ 7 \\ 5 \\ 8 \end{bmatrix}}_{\mathbf{s}}$$

- Hadamard product ($\odot$) simply performs element-wise multiplication
- vector $\boldsymbol{g}$ acts as a gate: masks out parts of the memory and input
- gating should depend on memory state and input; learn its behavior
- differentiable gates: assume $\boldsymbol{g} \in \mathbb{R}^n$ but squash into $(0, 1)$ with σ

Long Short-Term Memory RNNs (LSTMs)



- State vectors s_i partitioned into context memory and hidden state [Hochreiter and Schmidhuber, 1997];

Long Short-Term Memory RNNs (LSTMs)



- State vectors s_i partitioned into context memory and hidden state [Hochreiter and Schmidhuber, 1997];
- forget gate f: how much of the previous memory to keep;
- input gate i: how much of the proposed update to apply;
- output gate o: what parts of the updated memory to output.

Long Short-Term Memory RNNs (LSTMs)



- State vectors $\boldsymbol{s_i}$ partitioned into context memory and hidden state [Hochreiter and Schmidhuber, 1997];
- forget gate $\boldsymbol{f}$: how much of the previous memory to keep;
- input gate $\boldsymbol{i}$: how much of the proposed update to apply;
- output gate $\boldsymbol{o}$: what parts of the updated memory to output.

$$
\begin{aligned}
\boldsymbol{s_i} &= \mathrm{R}(\boldsymbol{x_i}, \boldsymbol{s_{i-1}}) = [\boldsymbol{c_i}; \boldsymbol{h_i}] \\
\boldsymbol{f} &= \sigma(\boldsymbol{x_i}\boldsymbol{W^{xf}} + \boldsymbol{h_{i-1}}\boldsymbol{W^{hf}}) \\
\boldsymbol{i} &= \sigma(\boldsymbol{x_i}\boldsymbol{W^{xi}} + \boldsymbol{h_{i-1}}\boldsymbol{W^{hi}}) \\
\boldsymbol{o} &= \sigma(\boldsymbol{x_i}\boldsymbol{W^{xo}} + \boldsymbol{h_{i-1}}\boldsymbol{W^{ho}}) \\
\boldsymbol{c_i} &= \boldsymbol{f} \odot \boldsymbol{c_{i-1}} + \boldsymbol{i} \odot \tanh(\boldsymbol{x_i}\boldsymbol{W^{x}} + \boldsymbol{h_{i-1}}\boldsymbol{W^{h}}) \\
\boldsymbol{h_i} &= \boldsymbol{o} \odot \tanh(\boldsymbol{c_i}) \\
\boldsymbol{y_i} &= O(\boldsymbol{s_i}) = \boldsymbol{h_i}
\end{aligned}
$$

Long Short-Term Memory RNNs (LSTMs)



- State vectors $\boldsymbol{s_i}$ partitioned into context memory and hidden state [Hochreiter and Schmidhuber, 1997];
- forget gate $\boldsymbol{f}$: how much of the previous memory to keep;
- input gate $\boldsymbol{i}$: how much of the proposed update to apply;
- output gate $\boldsymbol{o}$: what parts of the updated memory to output.

$$
\begin{aligned}
\boldsymbol{s_i} &= \mathrm{R}(\boldsymbol{x_i}, \boldsymbol{s_{i-1}}) = [\boldsymbol{c_i}; \boldsymbol{h_i}] \\
\boldsymbol{f} &= \sigma(\boldsymbol{x_i}\boldsymbol{W^{xf}} + \boldsymbol{h_{i-1}}\boldsymbol{W^{hf}}) \\
\boldsymbol{i} &= \sigma(\boldsymbol{x_i}\boldsymbol{W^{xi}} + \boldsymbol{h_{i-1}}\boldsymbol{W^{hi}}) \\
\boldsymbol{o} &= \sigma(\boldsymbol{x_i}\boldsymbol{W^{xo}} + \boldsymbol{h_{i-1}}\boldsymbol{W^{ho}}) \\
\boldsymbol{c_i} &= \boldsymbol{f} \odot \boldsymbol{c_{i-1}} + \boldsymbol{i} \odot \tanh(\boldsymbol{x_i}\boldsymbol{W^{x}} + \boldsymbol{h_{i-1}}\boldsymbol{W^{h}}) \\
\boldsymbol{h_i} &= \boldsymbol{o} \odot \tanh(\boldsymbol{c_i}) \\
\boldsymbol{y_i} &= O(\boldsymbol{s_i}) = \boldsymbol{h_i}
\end{aligned}
$$

- More parameters: separate $\boldsymbol{W^{x\cdot}}$ and $\boldsymbol{W^{h\cdot}}$ matrices for each gate.

Long Short-Term Memory RNNs (LSTMs)

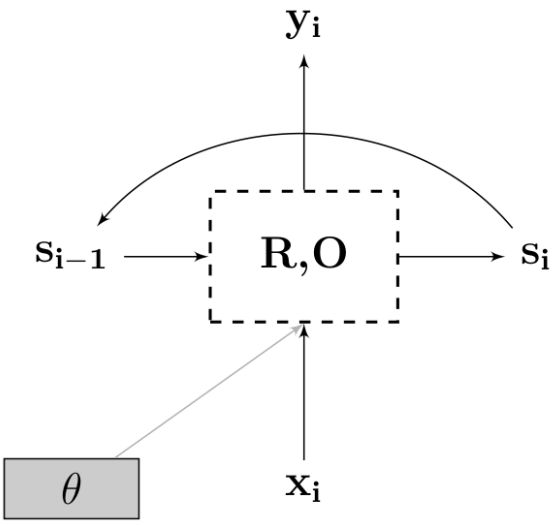


- State vectors s_i partitioned into context memory and hidden state [Hochreiter and Schmidhuber, 1997];
- forget gate f: how much of the previous memory to keep;
- input gate i: how much of the proposed update to apply;
- output gate o: what parts of the updated memory to output.

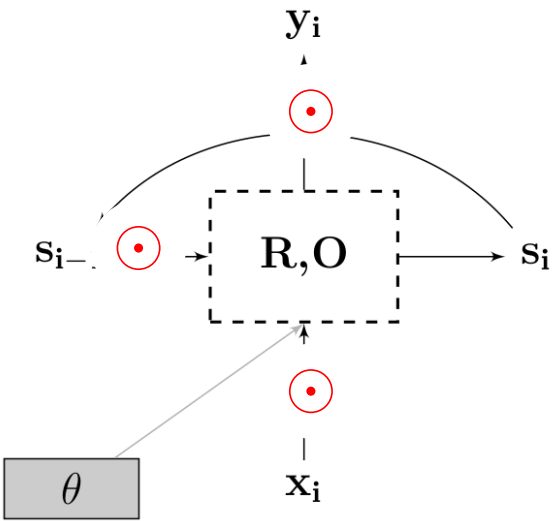
$$
\begin{aligned}
s_i &= \mathrm{R}(x_i, s_{i-1}) = [c_i; h_i] \\
f_i &= \sigma(x_i W^{xf} + h_{i-1} W^{hf} + b_f) \\
i_i &= \sigma(x_i W^{xi} + h_{i-1} W^{hi} + b_i) \\
o_i &= \sigma(x_i W^{xo} + h_{i-1} W^{ho} + b_o) \\
c_i &= f_i \odot c_{i-1} + i_i \odot \tanh(x_i W^{x} + h_{i-1} W^{h} + b) \\
h_i &= o_i \odot \tanh(c_i) \\
y_i &= O(s_i) = h_i
\end{aligned}
$$

- More parameters: separate $W^{x\cdot}$ and $W^{h\cdot}$ matrices for each gate.

Learning to Forget: Schematically



Learning to Forget: Schematically



Gated Recurrent Units (GRUs)



GRUs simplify the idea

- No separate context memory
- Less gates

Gated Recurrent Units (GRUs)

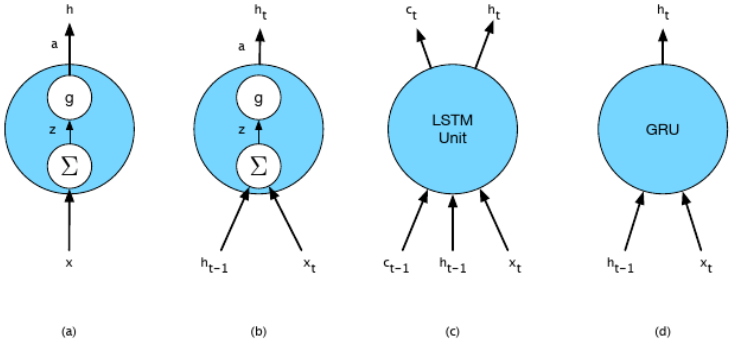
GRUs simplify the idea

- No separate context memory
- Less gates

$$
\begin{aligned}
\boldsymbol{s_j} &= \mathrm{R}_{\mathrm{GRU}}(\boldsymbol{s_{j-1}}, \boldsymbol{x_j}) = (1 - \boldsymbol{z}) \odot \boldsymbol{s_{j-1}} + \boldsymbol{z} \odot \tilde{\boldsymbol{s}}_{\boldsymbol{j}} \\
\boldsymbol{z} &= \sigma(\boldsymbol{x_j W^{xz}} + \boldsymbol{s_{j-1} W^{sz}}) \\
\boldsymbol{r} &= \sigma(\boldsymbol{x_j W^{xr}} + \boldsymbol{s_{j-1} W^{sr}}) \\
\tilde{\boldsymbol{s}}_{\boldsymbol{j}} &= \tanh(\boldsymbol{x_j W^{xs}} + (\boldsymbol{r} \odot \boldsymbol{s_{j-1}}) \boldsymbol{W^{sg}}) \\
\boldsymbol{y_j} &= O_{GRU}(\boldsymbol{s_j}) = \boldsymbol{s_j}
\end{aligned}
$$

[Cho et al., 2014]

Basic neural units



- a) feed-forward unit
- b) simple recurrent unit (Elman, etc)
- c) LSTM unit
- d) GRU unit

[Jurafsky and Martin 2021]

Streams of Fashion in NLP



To a first approximation, the de facto consensus in NLP in 2017 is that, no matter what the task, you throw a BiLSTM at it [...]

Christopher Manning, March 2017

(`https://simons.berkeley.edu/talks/christopher-manning-2017-3-27`)

Streams of Fashion in NLP



To a first approximation, the de facto consensus in NLP in 2017 is that, no matter what the task, you throw a BiLSTM at it [...]

Christopher Manning, March 2017

(`https://simons.berkeley.edu/talks/christopher-manning-2017-3-27`)

RNNs, particularly ones with gated architectures such as the LSTM and the GRU, [...] are arguably the strongest contribution of deep learning to the statistical natural language processing tool set.

[Goldberg, 2017]

Streams of Fashion in NLP



To a first approximation, the de facto consensus in NLP in 2017 is that, no matter what the task, you throw a BiLSTM at it [...]

Christopher Manning, March 2017

(`https://simons.berkeley.edu/talks/christopher-manning-2017-3-27`)

RNNs, particularly ones with gated architectures such as the LSTM and the GRU, [...] are arguably the strongest contribution of deep learning to the statistical natural language processing tool set.

[Goldberg, 2017]

But wait for Transformers and the 'Attention is all you need' motto of the last 3 years!
(later in the course)

Recap: RNNs In a Nutshell



- State vectors s_i reflect the complete history up to time point (word or character) i;
- RNNs are sensitive to basic natural language structures: sequences;

Recap: RNNs In a Nutshell



- State vectors $\boldsymbol{s_i}$ reflect the complete history up to time point (word or character) i;
- RNNs are sensitive to basic natural language structures: sequences;
- applicable to arbitrary length inputs (in principle);
- few parameters: weight matrices shared across all time points;

Recap: RNNs In a Nutshell



- State vectors s_i reflect the complete history up to time point (word or character) i;
- RNNs are sensitive to basic natural language structures: sequences;
- applicable to arbitrary length inputs (in principle);
- few parameters: weight matrices shared across all time points;
- analogous to (potentially) deep nesting: repeated multiplications;
- practical limitation: 'exploding' or 'vanishing' gradients;

Recap: RNNs In a Nutshell



- State vectors s_i reflect the complete history up to time point (word or character) i;
- RNNs are sensitive to basic natural language structures: sequences;
- applicable to arbitrary length inputs (in principle);
- few parameters: weight matrices shared across all time points;
- analogous to (potentially) deep nesting: repeated multiplications;
- practical limitation: 'exploding' or 'vanishing' gradients;
- Solution $\rightarrow$ gated RNNs: LSTM [Hochreiter and Schmidhuber, 1997] and GRU [Cho et al., 2014].

References I

Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (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), pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.

Goldberg, Y. (2017).
Neural network methods for natural language processing.
Synthesis Lectures on Human Language Technologies, 10(1):1–309.

Hochreiter, S. and Schmidhuber, J. (1997).
Long short-term memory.
Neural computation, 9(8):1735–1780.