RECURRENT NEURAL NETWORKS

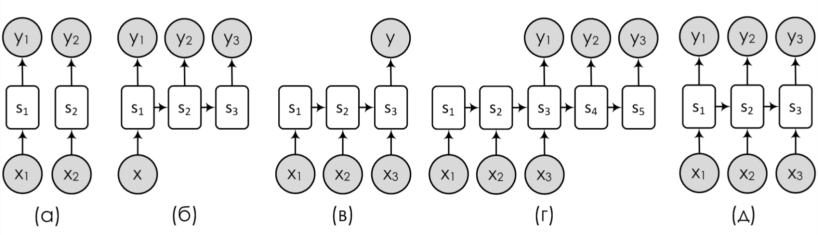
Master's Deep Learning

Sergey Nikolenko

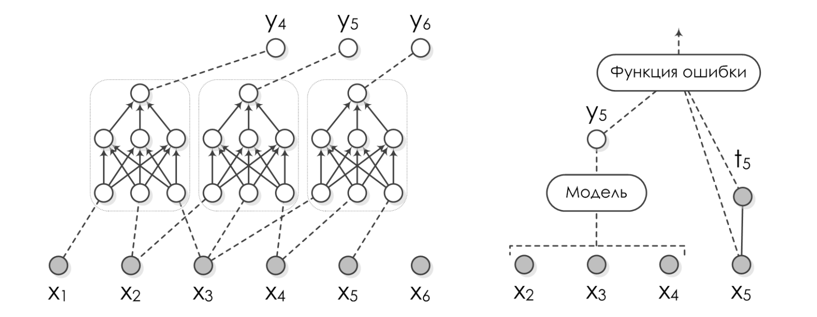
Harbour Space University, Barcelona, Spain
November 16, 2017

WORKING WITH SEQUENCES

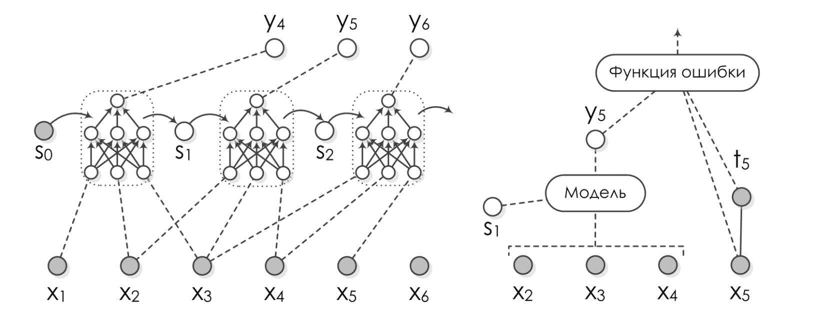
- Sequences: financial time series, language, sound...
- Different kinds of sequence-based problems:



- How to we apply a neural network to a sequence-based problem?
- We can try to use a sliding window:



- …but it would be better to have a hidden state and update it.
- This is the whole idea of *recurrent neural networks* (RNN).



- But how do we do backprop? Doesn't the computational graph have cycles now that

$$s_i = h(x_i, x_{i+1}, x_{i+2}, s_{i-1})?$$

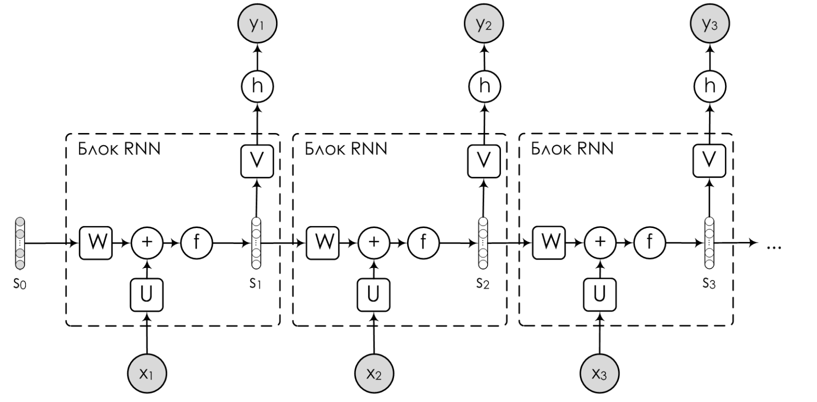
- …not really. We can "unroll" it back:

$$\begin{aligned} y_6 = f(x_3, x_4, x_5, s_2) = f(x_3, x_4, x_5, h(x_2, x_3, x_4, s_1)) = \\ = f(x_3, x_4, x_5, h(x_2, x_3, x_4, h(x_1, x_2, x_3, s_0))). \end{aligned}$$

- So formally there is no problem.
- But, of course, lots of practical issues – it's a *very* deep network with lots of shared weights…

RECURRENT NEURAL NETWORKS

- “Simple” RNN:


y1
y2
y3
h
h
h
Блок RNN
Блок RNN
Блок RNN
V
V
V
W
+
f
W
+
f
W
+
f
...
s0
s1
s2
s3
U
U
U
x1
x2
x3

- Formally:

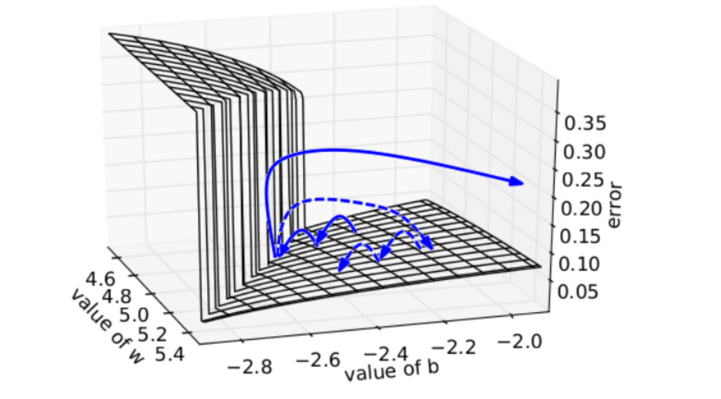
$$\begin{aligned}\mathbf{a}_t &= \mathbf{b} + W\mathbf{s}_{t-1} + U\mathbf{x}_t,\\ \mathbf{s}_t &= f(\mathbf{a}_t),\\ o_t &= c + V\mathbf{s}_t,\\ \mathbf{y}_t &= h(o_t),\end{aligned}$$

  where f is the recurrent nonlinearity, and h is the output function.
- Two problems:
  - exploding gradients;
  - vanishing gradients.

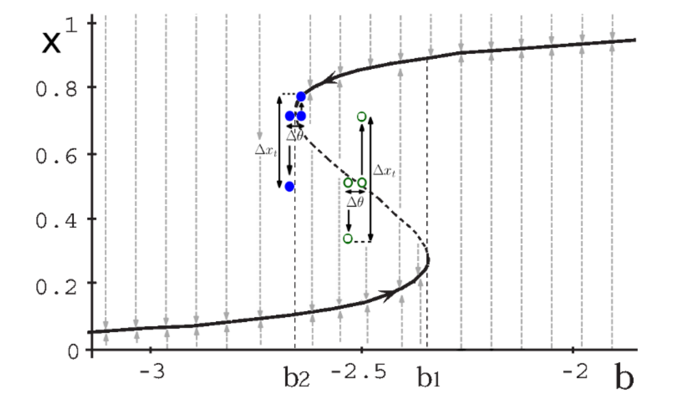
- We multiply by the same W, and the norm of the gradient grows or decreases exponentially.
- What do we do when it explodes?

- We can simply bound the gradients from above!
- Two versions — bound either the norm or every value:
    - `sgd = optimizers.SGD(lr=0.01, clipnorm=1.)`
    - `sgd = optimizers.SGD(lr=0.01, clipvalue=.05)`

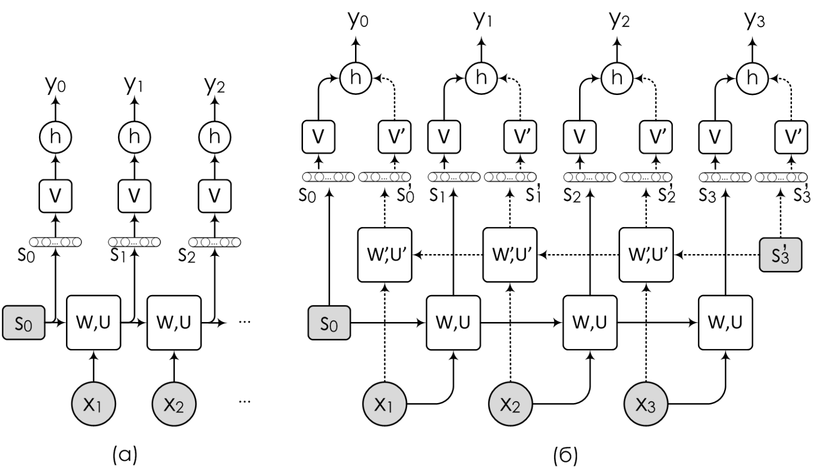
- (Pascanu et al., 2013) – nice pictures about it:



- Also explains where such gaps come from – RNNs have “bifurcation points”:



- Sometimes we need context from both directions:



- Formally:

$$\begin{aligned}
\mathbf{s}_t &= \sigma\left(\mathbf{b} + W\mathbf{s}_{t-1} + U\mathbf{x}_t\right), \\
\mathbf{s}'_t &= \sigma\left(\mathbf{b}' + W'\mathbf{s}'_{t+1} + U'\mathbf{x}_t\right), \\
o_t &= c + V\mathbf{s}_t + V'\mathbf{s}'_t, \\
\mathbf{y}_t &= h\left(o_t\right).
\end{aligned}$$

- This, of course, generalizes to any other sort of constructions.

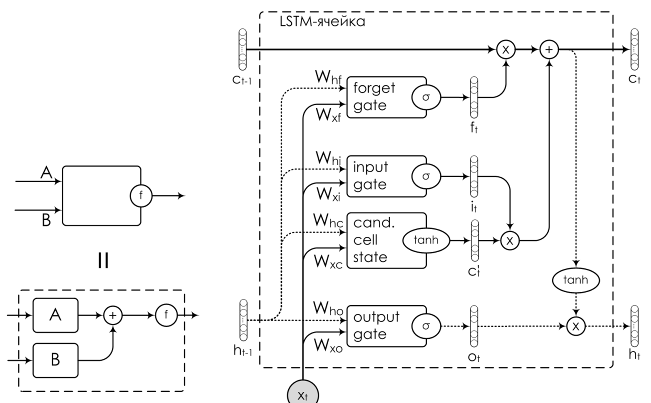
LSTM AND GRU

- Vanishing gradients: we have to multiply by W every time.
- This makes it impossible to have long-term memory.

- What do we do?..

- ...we have seen the basic idea in ResNet: we have to provide the "constant error carousel", a shortcut for the gradients.
- Idea from the mid-1990s (Schmidhuber): let's construct RNNs from a more complex unit that has the shortcut and controls memory explicitly.
- LSTM (long short-term memory).

- "Vanilla" LSTM: c_t is the cell state, and h_t is the hidden state.
- Input gate and forget gate control whether we change c_t to the new candidate cell state.



(а) (б)

- Formally:

$$
\begin{aligned}
c'_t &= \tanh\left(W_{xc}\mathbf{x}_t + W_{hc}h_{t-1} + \mathbf{b}_{c'}\right) & \textit{candidate cell state} \\
i_t &= \sigma\left(W_{xi}\mathbf{x}_t + W_{hi}h_{t-1} + \mathbf{b}_i\right) & \textit{input gate} \\
f_t &= \sigma\left(W_{xf}\mathbf{x}_t + W_{hf}h_{t-1} + \mathbf{b}_f\right) & \textit{forget gate} \\
o_t &= \sigma\left(W_{xo}\mathbf{x}_t + W_{ho}h_{t-1} + \mathbf{b}_o\right) & \textit{output gate} \\
c_t &= f_t \odot c_{t-1} + i_t \odot c'_t, & \textit{cell state} \\
h_t &= o_t \odot \tanh(c_t) & \textit{block output}
\end{aligned}
$$

- So the LSTM cell is able to control the cell state with the hidden state and weights.
- Very flexible, and if the forget gate is closed ($f_t = 1$), we have

$$c_t = c_{t-1} + i_t \odot c'_t, \text{ so } \frac{\partial c_t}{\partial c_{t-1}} = 1.$$

- The constant error carousel!

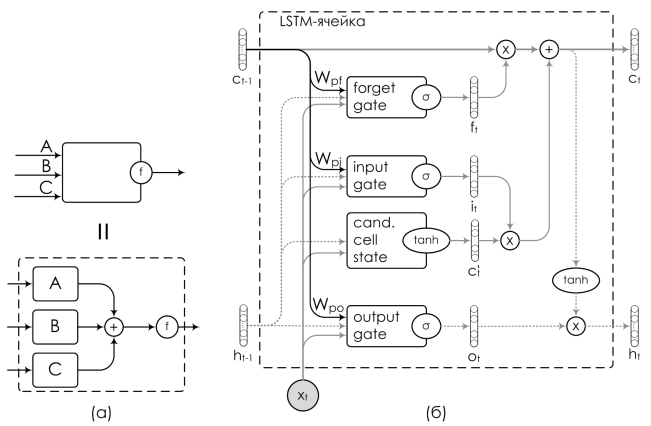
- LSTM was developed in mid-1990s (Hochreiter and Schmidhuber, 1995; 1997).
- Completely in its modern form in (Gers, Schmidhuber, 2000).
- One problem: we want to control c, but it's unavailable for the gates! They only see h_{t-1}, which is

$$h_{t-1} = o_{t-1} \odot \tanh(c_{t-1}).$$

- So if the output gate is closed the behaviour of LSTM does not depend on cell state at all.
- This is not good...

LSTM

- ...so of course we add a few more weight matrices (peepholes).



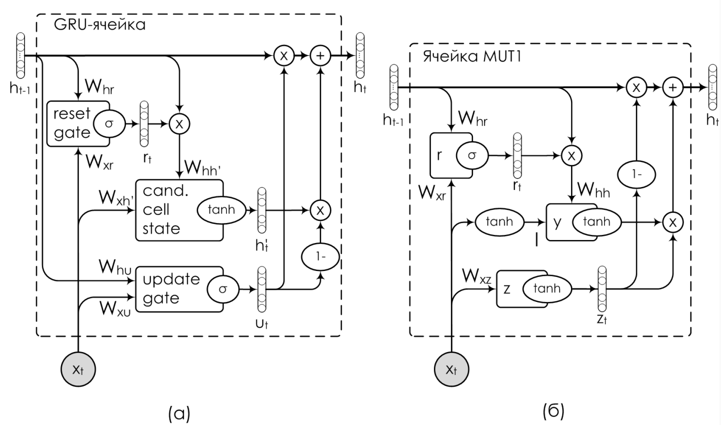
- Formally:

$$
\begin{aligned}
i_t &= \sigma\left(W_{xi}\mathbf{x}_t + W_{hi}h_{t-1} + W_{pi}c_{t-1} + \mathbf{b}_i\right) \\
f_t &= \sigma\left(W_{xf}\mathbf{x}_t + W_{hf}h_{t-1} + W_{pf}c_{t-1} + \mathbf{b}_f\right) \\
o_t &= \sigma\left(W_{xo}\mathbf{x}_t + W_{ho}h_{t-1} + W_{po}c_{t-1} + \mathbf{b}_o\right)
\end{aligned}
$$

- Huge number of LSTM variations: we can basically remove any kind of gate, add or remove each peephole, add or remove activation functions etc.
- How do we choose?

- «LSTM: a Search Space Odyssey» (Greff et al., 2015).
- Shows an experimental comparison of many variations.
- In particular, some significantly simpler architectures (with one less gate!) did not lose to the vanilla LSTM much.
- This brings us to…

- …Gated Recurrent Units (GRU); developed in (Cho et al., 2014).
- Also implements the constant error carousel, but simpler than LSTM.

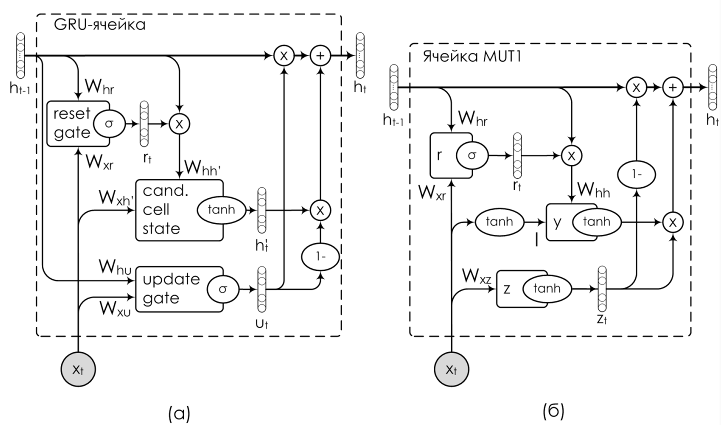


- Formally:

$$
\begin{aligned}
u_t &= \sigma(W_{xu}\mathbf{x}_t + W_{hu}h_{t-1} + \mathbf{b}_u) \\
r_t &= \sigma(W_{xr}\mathbf{x}_t + W_{hr}h_{t-1} + \mathbf{b}_r) \\
h'_t &= \tanh(W_{xh'}\mathbf{x}_t + W_{hh'}(r_t \odot h_{t-1})) \\
h_t &= (1 - u_t) \odot h'_t + u_t \odot h_{t-1}
\end{aligned}
$$

- Update gate and reset gate; there is no distinction between c_t and h_t.
- Fewer matrices (6 vs. 8 or 11 with peepholes), fewer weights, but only very slightly worse than LSTM.
- So you can fit more GRUs and have better networks on a given computational budget.

- There are other variations too.
- (Józefowicz, Zaremba, Sutskever, 2015): huge experimental comparison, with an evolutionary approach.
- Identified three interesting architectures.



THANK YOU!

Thank you for your attention!