LSTM compression for language modeling

Artem Grachev, Svyatoslav Elizarov, Boris Kovalenko

Higher School of Economics

May 19th, 2018

Outline

Language modeling

We need to compute the probability of a sentence or sequence of words $(w_1, \ldots, w_T)$ in language L.

$$\mathrm{P}(w_1, \ldots, w_T) = \mathrm{P}(w_1, \ldots, w_{T-1})\,\mathrm{P}(w_T | w_1, \ldots, w_{T-1}) =$$
$$= \prod_{t=1}^{T} \mathrm{P}(w_t | w_1, \ldots, w_{t-1})$$

Fix N and try to compute $\mathrm{P}(w_t | w_{t-N}, \ldots, w_{t-1})$

RNN

RNN – recurrent neural network. T – number timesteps, L – number recurrent layers. Each layer we can describe as follows:

$$z_l^t = W_i x_{l-1}^t + V_l x_l^{t-1} + b_l \tag{1}$$
$$x_l^t = \sigma(z_l) \tag{2}$$

$t \in 1, \ldots, N$; $l \in 1, \ldots, L$. The output of the network is given by

$$y^t = \text{softmax}\left[W_{L+1} x_L^t + b_t^{L+1}\right]$$

Then define

$$\text{P}\left(w_t | w_{t-N}, \ldots, w_{t-1}\right) = y^t$$

LSTM

Equations for one LSTM-layer:

$$i_l^t = \sigma\left[W_l^i x_{l-1}^t + V_l^i x_l^{t-1} + b_l^i + D(v_l^i)c_l^{t-1}\right] \quad \text{input gate} \quad (3)$$

$$f_l^t = \sigma\left[W_l^f x_{l-1}^t + V_l^f x_l^{t-1} + b_l^f + D(v_l^f)c_l^{t-1}\right] \quad \text{forget gate} \quad (4)$$

$$c_l^t = f_l^t \cdot c_l^{t-1} + i_l^t \tanh\left[W_l^c x_{l-1}^t + U_l^c x_l^{t-1} + b_l^c\right] \quad \text{cell state} \quad (5)$$

$$o_l^t = \sigma\left[W_l^o x_{l-1}^t + V_l^o x_l^{t-1} + b_l^o + D(v_l^o)c_l^{t-1}\right] \quad \text{output gate} \quad (6)$$

$$x_l^t = o_l^t \cdot \tanh[c_l^t] \quad (7)$$

$t \in 1, \ldots, N$; $l \in 1, \ldots, L$. The output of the network is given by

$$y^t = \text{softmax}\left[W_{L+1} x_L^t + b_t^{L+1}\right]$$

LSTM unit

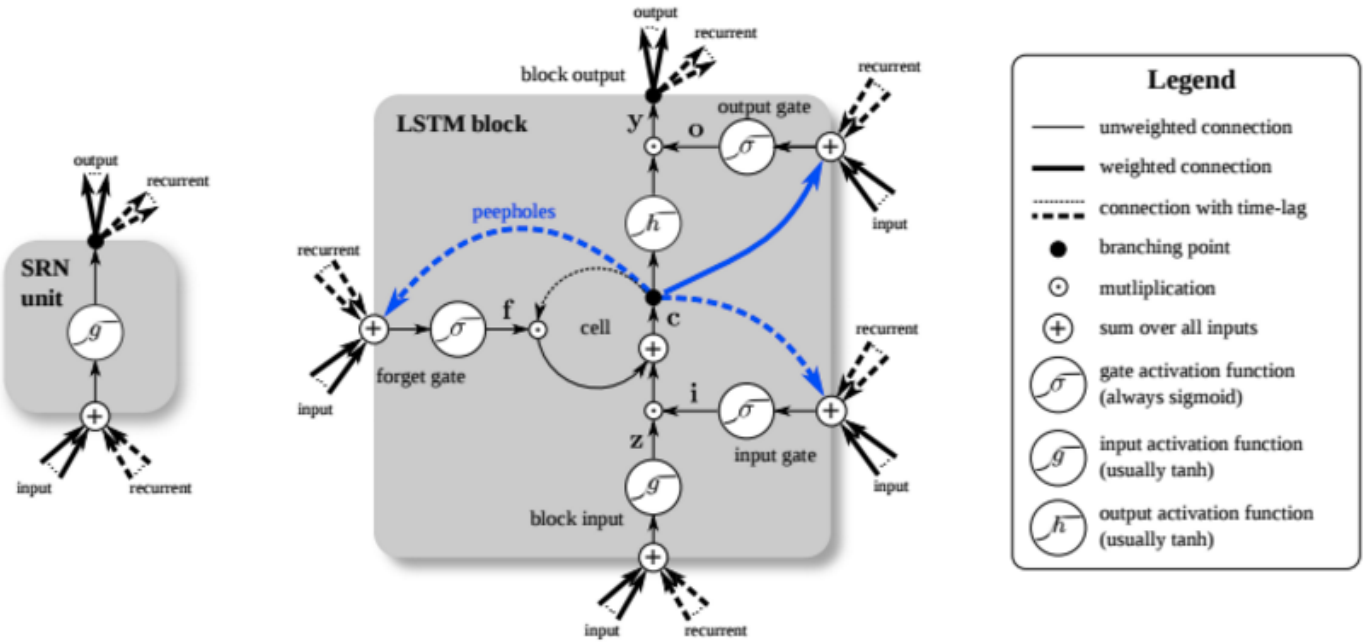


Figure 1: Left: RNN unit, Right: LSTM unit

Problem

Neural network models can take up a lot of space on disk and in memory. Also they need a lot of time for inference.

Let's compress neural networks

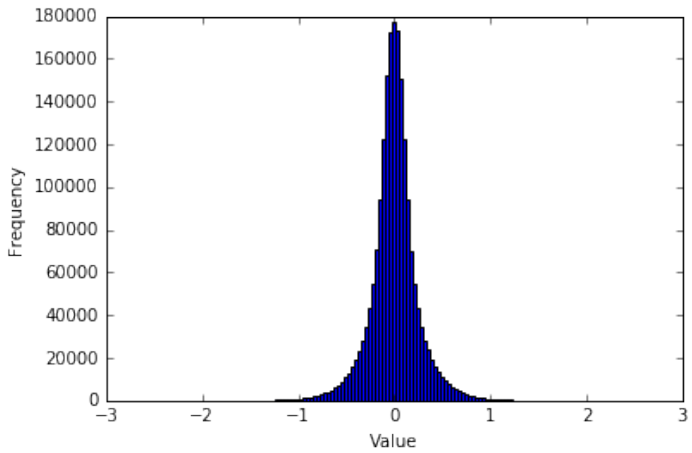
Outline

1. Pruning
2. Quantization
3. Low-rank decomposition

Pruning

Definition

Pruning – method for reducing number of parameters in NN. All connections with weights below a threshold are removed from the network. After this we retrain the network to learn the final weights for the remaining sparse connections



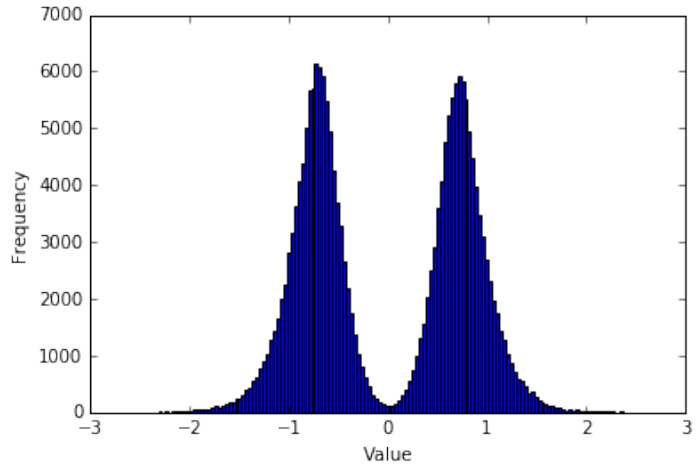


Figure 2: Weights distribution before and after pruning

Quantization[1]

Definition

Quantization – method for compression the size of neural network in memory. We are compressing each float value to an eight-bit integer representing the closest real number in a linear set of 256 within the range.

When we use the model for inference we still need to do full-precision computations.

[1] https://petewarden.com/2016/05/03/how-to-quantize-neural-networks-with-tensorflow/

PTB Results

Method	Size (Mb)	Test PP
Original. 2-layer (each 200) LSTM.	18.6 Mb	117.659
Pruning output layer 90% w/o additional training	5.5 Mb	149.310
Pruning output layer 90% with additional training	5.5 Mb	121.123
Quantization.	4.7 Mb	118.232

Table 1: Pruning and quantization results

Low-rank factorization

Simple factorization can be done as follows:

$$x_l^t = \sigma \left[W_l^a W_l^b h_{l-1}^t + U_l^a U_l^b h_l^{t-1} + b_l \right]$$

Let's require $W_l^b = U_{l-1}^b$. After this we can rewrite our equation for RNN:

$$x_l^t = \sigma \left[W_l^a m_{l-1}^t + U_l^a m_l^{t-1} + b_l \right] \tag{8}$$

$$m_l^t = U_l^b x_l^t \tag{9}$$

$$y_t = softmax \left[W_{L+1} m_L^t + b_{L+1} \right] \tag{10}$$

Note that for LSTM it is pretty the same

Low-rank factorization. Visual results

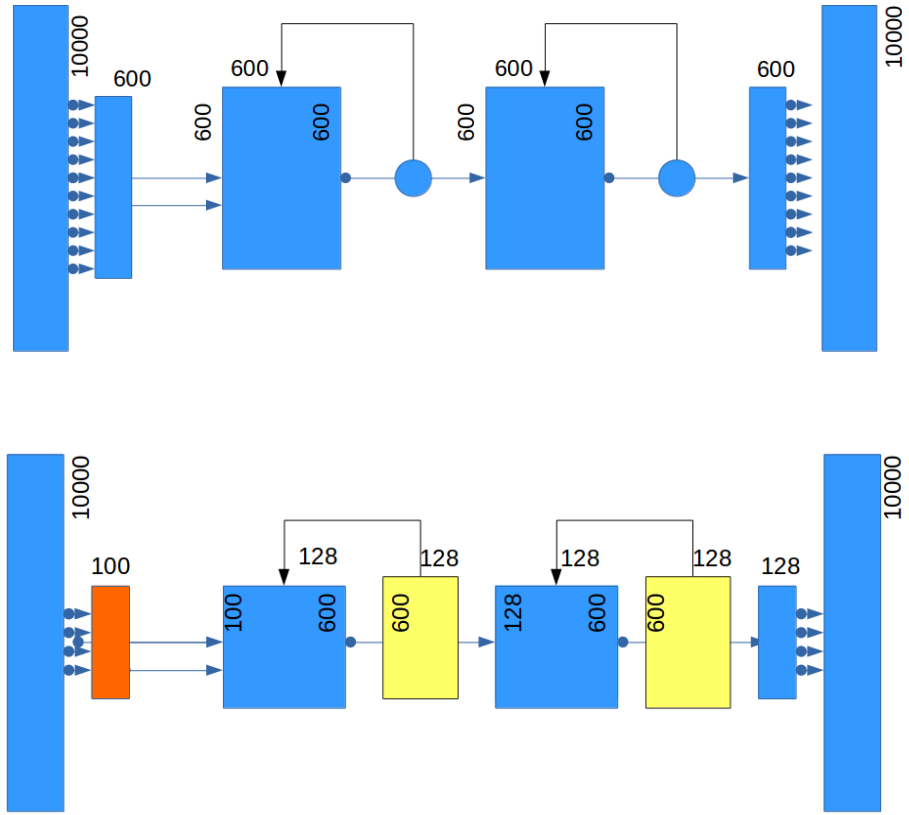


Figure 3: up — original, down — compressed

Low-rank factorization. Results

Method	**Size (Mb)**	**Test PP**
Plain vanilla model 2-layer (each 600) LSTM	71.1	43.4
SVD for output layer (128)	52.5	43.6
Reduce embedding size (100)	46.3	42.9
Reduce embedding size (100) and SVD for output layer (128)	27.7	42.9
Reduce embedding size (100) and Low-rank factorization (128)	14.5	45.0
Reduce embedding size (100) and Low-rank factorization (128) in half-precision	7.3	45.0

Table 2: Low-rank factorization results. (not PTB)

Outline

What is a tensor[2] ?

Tensor = multidimensional array:

$$\vec{A} = [A(i_1, \ldots, i_d)], \qquad i_k \in \{1, \ldots, n_k\}. \tag{11}$$

Terminology:

- *dimensionality* $= d$ (number of indices).
- *size* $= n_1 \times \cdots \times n_d$ (number of nodes along each axis).

Case $d = 1 \Rightarrow$ vector, $d = 2 \Rightarrow$ matrix.

[2] The whole tensor-train part of this presentation was made by Anton Rodomanov (bayesgroup.ru)

Curse of dimensionality

Number of elements $= n^d$ (exponential in d)
When $n = 2,\ d = 100$

$$2^{100} > 10^{30} \qquad (\approx 10^{18} \text{ PB of memory}). \tag{12}$$

Cannot work with tensors using standard methods.

Tensor rank decomposition [Hitchcock, 1927]

Recall the rank decomposition for matrices:

$$A(i_1, i_2) = \sum_{\alpha=1}^{r} U(i_1, \alpha) V(i_2, \alpha). \tag{13}$$

This can be generalized to tensors.

Tensor rank decomposition (canonical decomposition):

$$A(i_1, \ldots, i_d) = \sum_{\alpha=1}^{R} U_1(i_1, \alpha) \ldots U_d(i_d, \alpha). \tag{14}$$

The minimal possible R is called the (canonical) rank of the tensor $\vec{A}$.

- (+) No curse of dimensionality.
- (-) Rank R should be known in advance for many methods.
- (-) Computation of R is NP-hard [Hillar, Lim, 2013].

Unfolding matrices: definition

Every tensor $\vec{A}$ has $d-1$ unfolding matrices:

$$A_k := [A(i_1 \dots i_k; i_{k+1} \dots i_d)], \tag{15}$$

where

$$A(i_1 \dots i_k; i_{k+1} \dots i_d) := A(i_1, \dots, i_d). \tag{16}$$

Here $i_1 \dots i_k$ and $i_{k+1} \dots i_d$ are row and column (multi)indices; A_k are matrices of size $M_k \times N_k$ with $M_k = \prod_{s=1}^{k} n_s$, $N_k = \prod_{s=k+1}^{d} n_s$.

This is just a `reshape`.

Unfolding matrices: example

Consider $\vec{A} = [A(i, j, k)]$ given by its elements:

$$\begin{aligned} A(1,1,1) &= 111, \quad A(2,1,1) = 211, \\ A(1,2,1) &= 121, \quad A(2,2,1) = 221, \\ A(1,1,2) &= 112, \quad A(2,1,2) = 212, \\ A(1,2,2) &= 122, \quad A(2,2,2) = 222. \end{aligned} \tag{17}$$

Then

$$A_1 = [A(i; jk)] = \begin{bmatrix} 111 & 121 & 112 & 122 \\ 211 & 221 & 212 & 222 \end{bmatrix}, \tag{18}$$

$$A_2 = [A(ij; k)] = \begin{bmatrix} 111 & 112 \\ 211 & 212 \\ 121 & 122 \\ 221 & 222 \end{bmatrix}. \tag{19}$$

Tensor Train decompositon: motivation

Main idea: variable splitting.

Consider a rank decomposition of an unfolding matrix:

$$A(i_1 i_2; i_3 i_4 i_5 i_6) = \sum_{\alpha_2} U(i_1 i_2; \alpha_2) V(i_3 i_4 i_5 i_6; \alpha_2). \tag{20}$$

On the left: 6-dimensional tensor; on the right: 3- and 5-dimensional.
The dimension has reduced!
Proceed recursively.

Tensor Train decomposition [Oseledets, 2011]

- TT-format for a tensor $\vec{A}$:

$$A(i_1, \ldots, i_d) = \sum_{\alpha_0, \ldots, \alpha_d} G_1(\alpha_0, i_1, \alpha_1) G_2(\alpha_1, i_2, \alpha_2) \ldots G_d(\alpha_{d-1}, i_d, \alpha_d). \tag{21}$$

- This can be written compactly as a matrix product:

$$A(i_1, \ldots, i_d) = \underbrace{G_1[i_1]}_{1 \times r_1} \underbrace{G_2[i_2]}_{r_1 \times r_2} \ldots \underbrace{G_d[i_d]}_{r_{d-1} \times 1} \tag{22}$$

- Terminology:
  - G_i: TT-cores (collections of matrices)
  - r_i: TT-ranks
  - $r = \max r_i$: maximal TT-rank
- TT-format uses $O(dnr^2)$ memory to store $O(n^d)$ elements.
- Efficient only if the ranks are small.

TT-format: example

- Consider a tensor:

$$A(i_1, i_2, i_3) := i_1 + i_2 + i_3, \tag{23}$$

$$i_1 \in \{1, 2, 3\}, \quad i_2 \in \{1, 2, 3, 4\}, \quad i_3 \in \{1, 2, 3, 4, 5\}. \tag{24}$$

- Its TT-format:

$$A(i_1, i_2, i_3) = G_1[i_1] G_2[i_2] G_3[i_3],$$

where

$$G_1[i_1] := \begin{bmatrix} i_1 & 1 \end{bmatrix}, \quad G_2[i_2] := \begin{bmatrix} 1 & 0 \\ i_2 & 1 \end{bmatrix}, \quad G_3[i_3] := \begin{bmatrix} 1 \\ i_3 \end{bmatrix}$$

- Check:

$$A(i_1, i_2, i_3) = \begin{bmatrix} i_1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ i_2 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ i_3 \end{bmatrix} =$$

$$= \begin{bmatrix} i_1 + i_2 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ i_3 \end{bmatrix} = i_1 + i_2 + i_3.$$

TT-format: example

- Consider a tensor:

$$A(i_1, i_2, i_3) := i_1 + i_2 + i_3, \tag{23}$$

$$i_1 \in \{1, 2, 3\}, \quad i_2 \in \{1, 2, 3, 4\}, \quad i_3 \in \{1, 2, 3, 4, 5\}. \tag{24}$$

- Its TT-format:

$$A(i_1, i_2, i_3) = G_1[i_1] G_2[i_2] G_3[i_3],$$

where

$$G_1 = \left(\begin{bmatrix} 1 & 1 \end{bmatrix}, \begin{bmatrix} 2 & 1 \end{bmatrix}, \begin{bmatrix} 3 & 1 \end{bmatrix} \right)$$

$$G_2 = \left(\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 2 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 3 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 4 & 1 \end{bmatrix} \right) \tag{25}$$

$$G_3 = \left(\begin{bmatrix} 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \begin{bmatrix} 1 \\ 3 \end{bmatrix}, \begin{bmatrix} 1 \\ 4 \end{bmatrix}, \begin{bmatrix} 1 \\ 5 \end{bmatrix} \right)$$

- The tensor has $3 \cdot 4 \cdot 5 = 60$ elements.
  TT-format uses 32 elements to describe it.

Finding a TT-representation of a tensor

General ways of building a TT-decomposition of a tensor:

- Analytical formulas for the TT-cores.
- TT-SVD algorithm [Oseledets, 2011]:
  - Exact quasi-optimal method.
  - Suitable only for small tensors (which fit into memory).
- Interpolation algorithms: AMEn-cross [Dolgov & Savostyanov, 2013], DMRG [Khoromskij & Oseledets, 2010], TT-cross [Oseledets, 2010]
  - Approximate heuristically-based methods.
  - Can be applied for large tensors.
  - No strong guarantees but work well in practice.
- Operations between other tensors in the TT-format: addition, element-wise product etc.

TT format. Summary

- TT-decomposition and corresponding algorithms are a good way to work with huge tensors.
- Memory and complexity depend on d linearly $\Rightarrow$ no curse of dimensionality.
- TT-format is efficient only if the TT-ranks are small. This is the case in many applications.
- Code is available:
  - Python: `https://github.com/oseledets/ttpy`
  - MATLAB: `https://github.com/oseledets/TT-Toolbox`

Results

Table 3: Matrix decomposition results on PTB dataset

	Model	Size	No. of params	Test PP
PTB Benchmarks	LSTM 200-200	18.6 Mb	4.64 M	117.659
	LSTM 650-650	79.1 Mb	19.7 M	82.07
	LSTM 1500-1500	264.1 Mb	66.02 M	78.29
Ours	LR LSTM 650-650	16.8 Mb	4.2 M	92.885
	TT LSTM 600-600	50.4 Mb	12.6 M	168.639
	LR LSTM 1500-1500	94.9 Mb	23.72 M	89.462

Outline

Future work

1. Better TT for LSTM (in proccess)
2. Variational dropout)

Thank you for listening!