# LSTM compression for language modeling

Artem Grachev

CS HSE, Samsung R&D Russia

April 27th, 2017

# 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

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

# 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

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

# RNN and LSTM unit

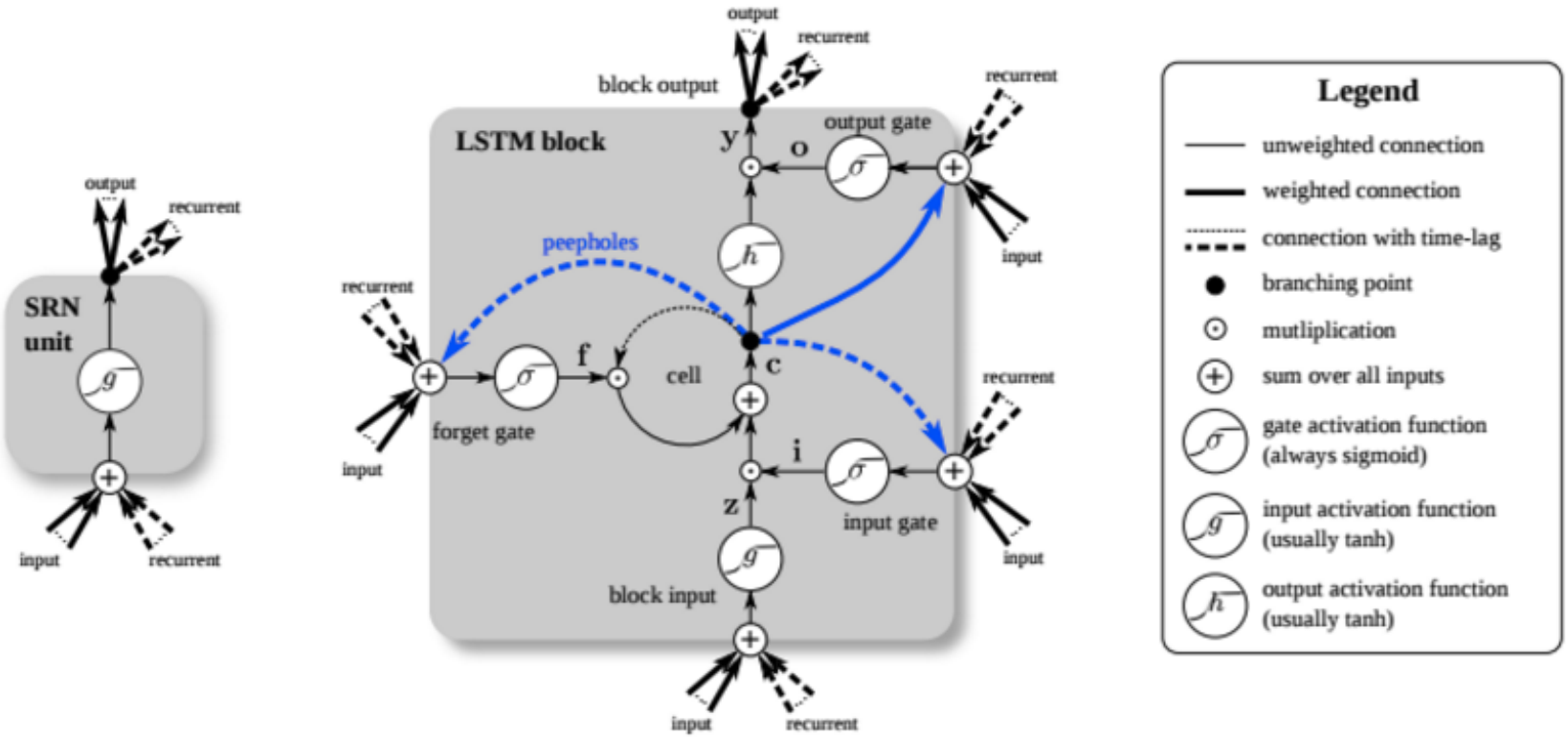



Figure 1: Left: RNN unit, Right: LSTM unit

## 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

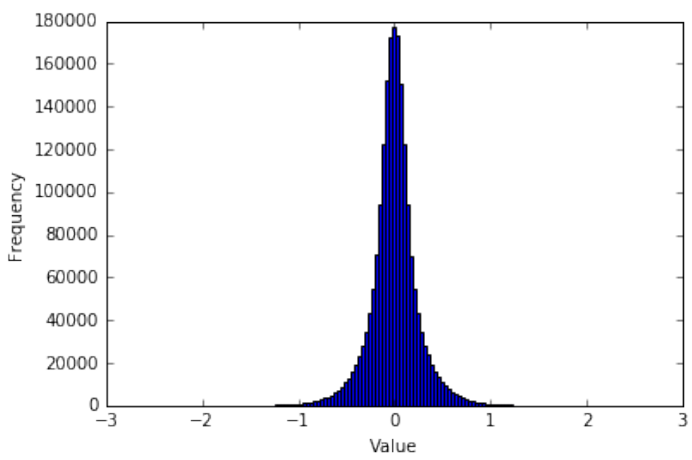
# Compression methods

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



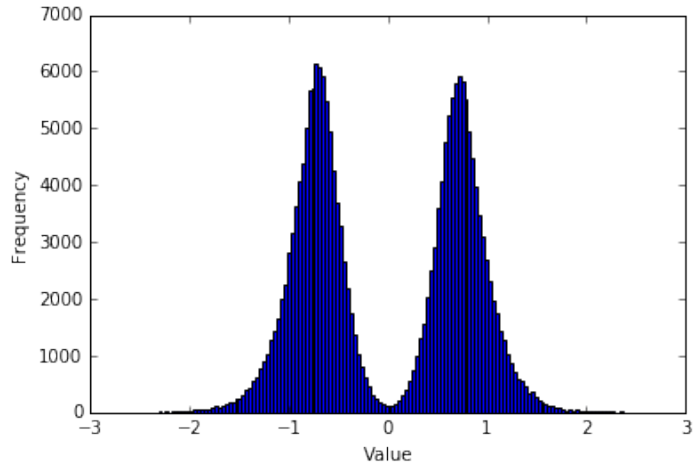




Figure 2: Weights distribution before and after pruning

# 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

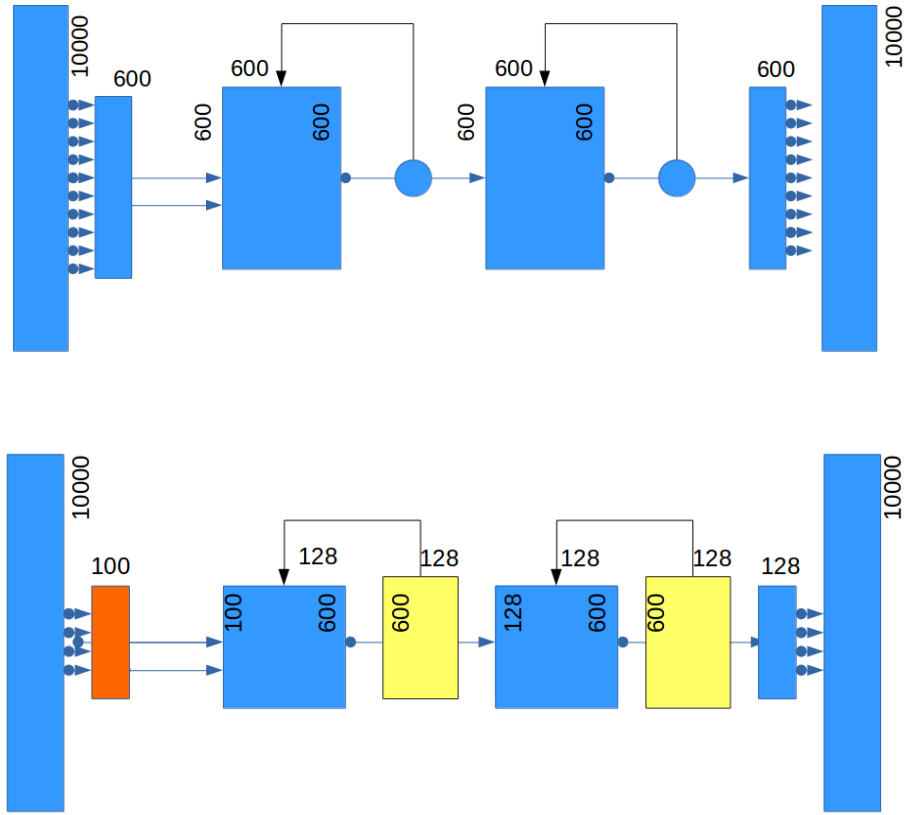



Figure 3: up — original, down — compressed

# Low-rank factorization. Results

| **Method** | **Size (Mb)** | **Test PP** |
|---|---|---|
| Plain vanilla model<br>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<br>SVD for output layer (128) | 27.7 | 42.9 |
| Reduce embedding size (100) and<br>Low-rank factorization (128) | 14.5 | 45.0 |
| Reduce embedding size (100) and<br>Low-rank factorization (128) in half-precision | 7.3 | 45.0 |

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

# Tensor Train decomposition [Oseledets, 2011]

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

$$A(i_1, \ldots, i_d) = \sum_{\alpha_0, \ldots, \alpha_d} G_1(\alpha_0, i_1, \alpha_1) G_2(\alpha_1, i_2, \alpha_2) \ldots G_d(\alpha_{d-1}, i_d, \alpha_d). \tag{11}$$

- This can be written compactly as a matrix product:

$$A(i_1, \ldots, i_d) = \underbrace{G_1[i_1]}_{1 \times r_1} \underbrace{G_2[i_2]}_{r_1 \times r_2} \ldots \underbrace{G_d[i_d]}_{r_{d-1} \times 1} \tag{12}$$

- 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.

# PTB results

| Method | Size (Mb) | Test PP |
|---|---|---|
| PTB 200-200. | 18.6 Mb | 115.91 |
| PTB 650-650 | 79.1 Mb | 82.07 |
| PTB 1500-1500 | 264.1 Mb | 78.29 |
| LR PTB 650-650 | 16.8 Mb | 92.885 |
| LR PTB 1500-1500 | 94.9 Mb | 89.462 |
| TT-LSTM PTB 600-600 | 50.4 Mb | 168.639 |

Table 3: PTB matrix decomposition results

# References



*Song Han and Huizi Mao and William J. Dally* Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding — Acoustics, Speech and Signal Processing (ICASSP), 2016


*Zhiyun Lu and Vikas Sindhwan and Tara N. Sainath* Learning compact recurrent neural networks — Acoustics, Speech and Signal Processing (ICASSP), 2016


*Y. Bengio.* Learning Deep Architectures for AI — Foundations and Trends in Machine Learning 2009


*Ivan V. Oseledets* Tensor-Train Decomposition — SIAM J. Scientific Computing Vol. 33(5), pp. 2295–2317., 2011


*Sepp Hochreiter and Jürgen Schmidhuber* Long Short-Term Memor — Neural Computation, Vol. 9(8), pp 1735-1780., 1997


*Tomáš Mikolov* Statistical Language Models Based on Neural Networks — Phd Thesis, Brno, CZ, 2012

Thank you for listening!