Higher School of Economics

Networks compression

Pruning - Quantization - Matrix factorization

Колесников Георгий

2020

Pruning
Quantization
Matrix factorization

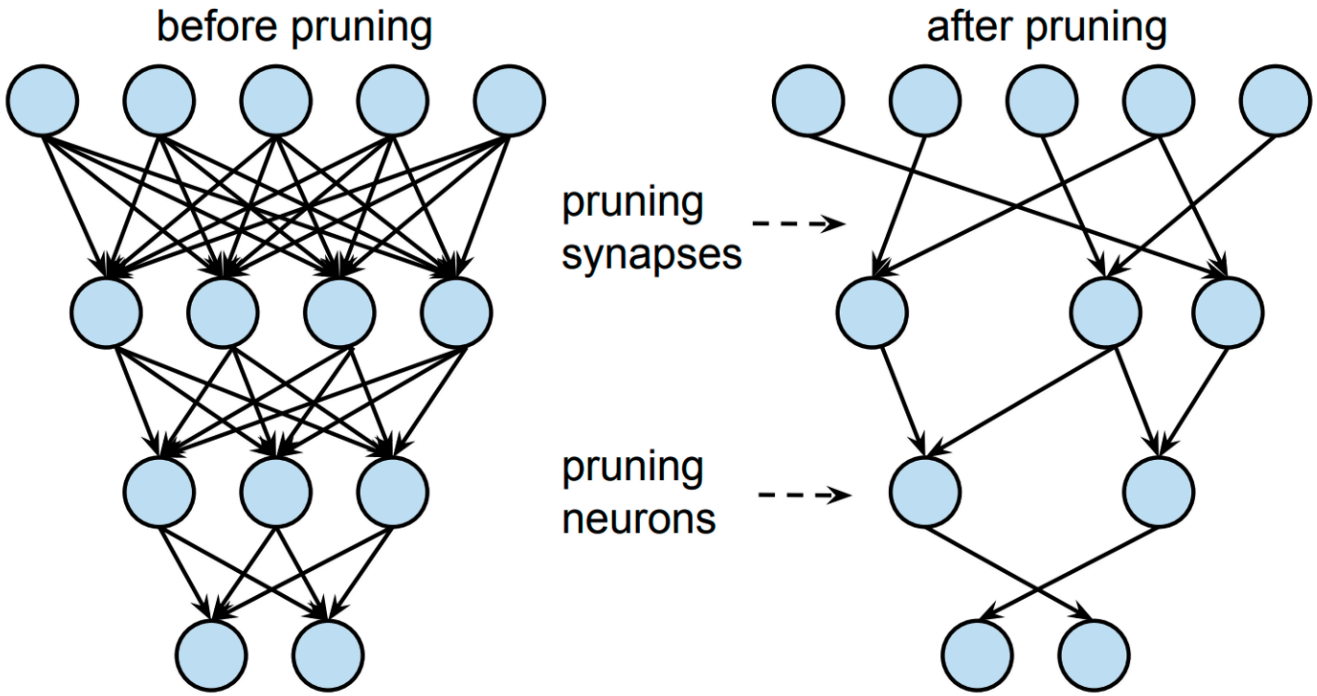
Проблема: большие нейронные сети, большие затраты памяти, больше времени работы и трата энергии

Решение: сжать нейронную сеть

Итог: более эффективные модели меньшего размера, меньше затраты энергии и минимум потери точности

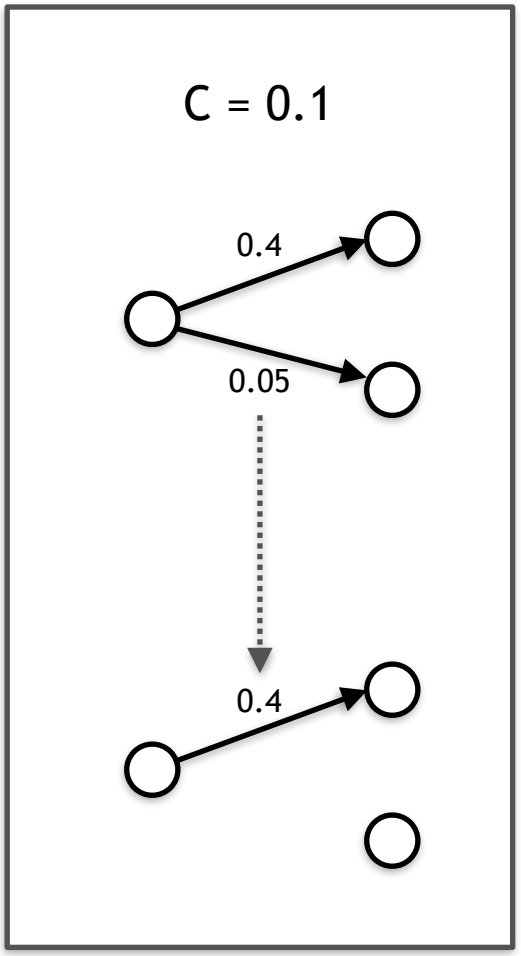
Pruning

- Обнуление весов (удаление синапсов)
- Удаление целых нейронов



Удаление синапсов

Удаляем веса близкие к нулю

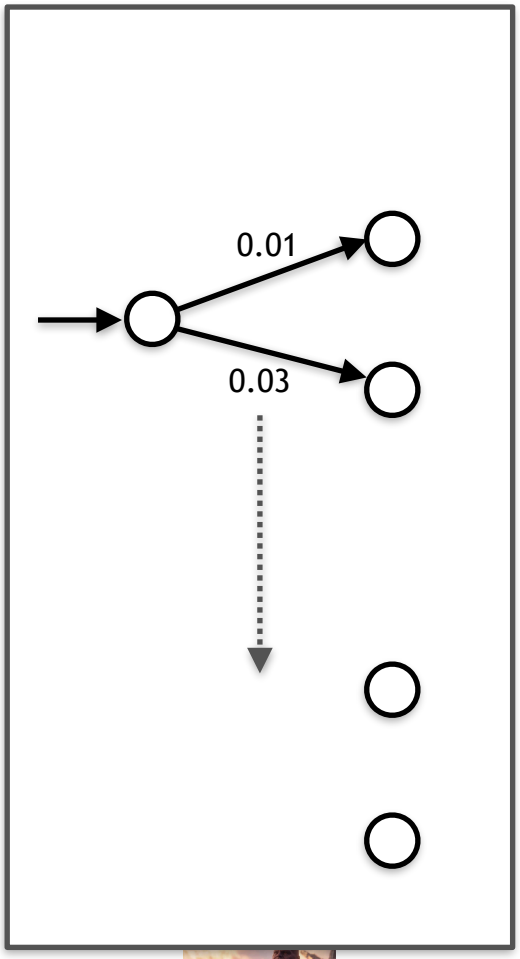


C - некоторый порог

Удаляем все w, такие что:

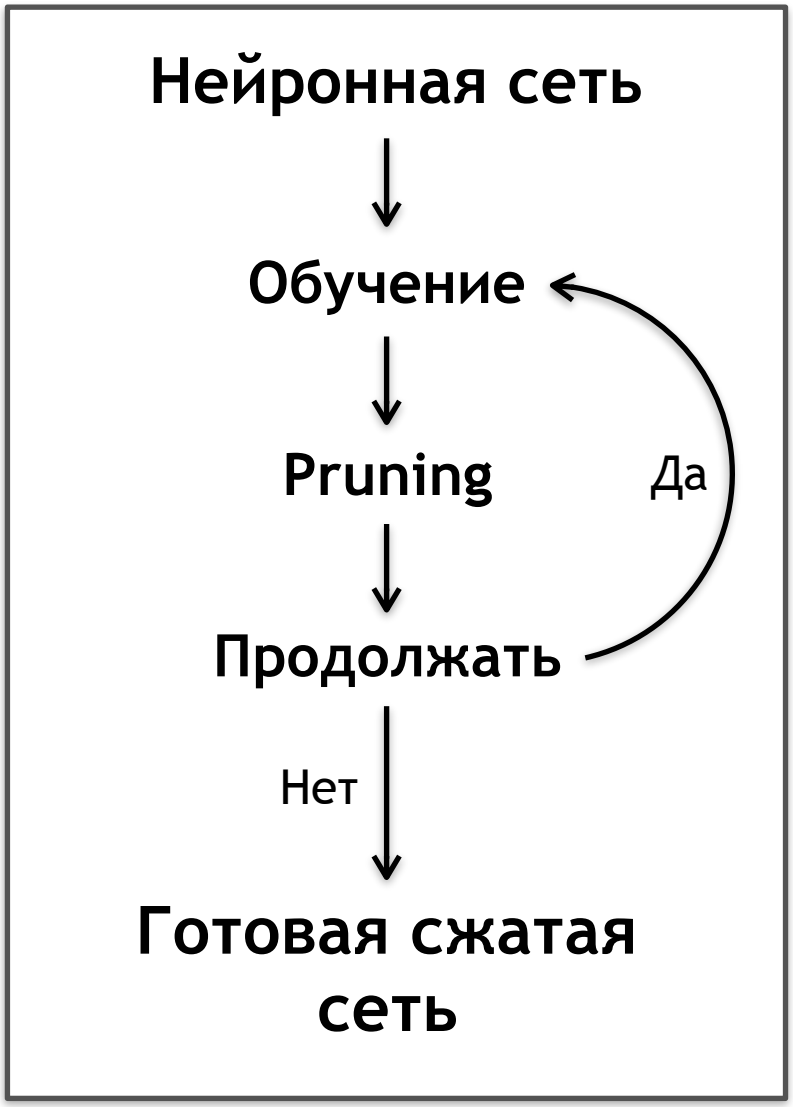
$$w < C$$

Удаление нейронов



Удаление нейронов которые редко выдают большие значение или дублирующие другие (выдающие близкие значения)

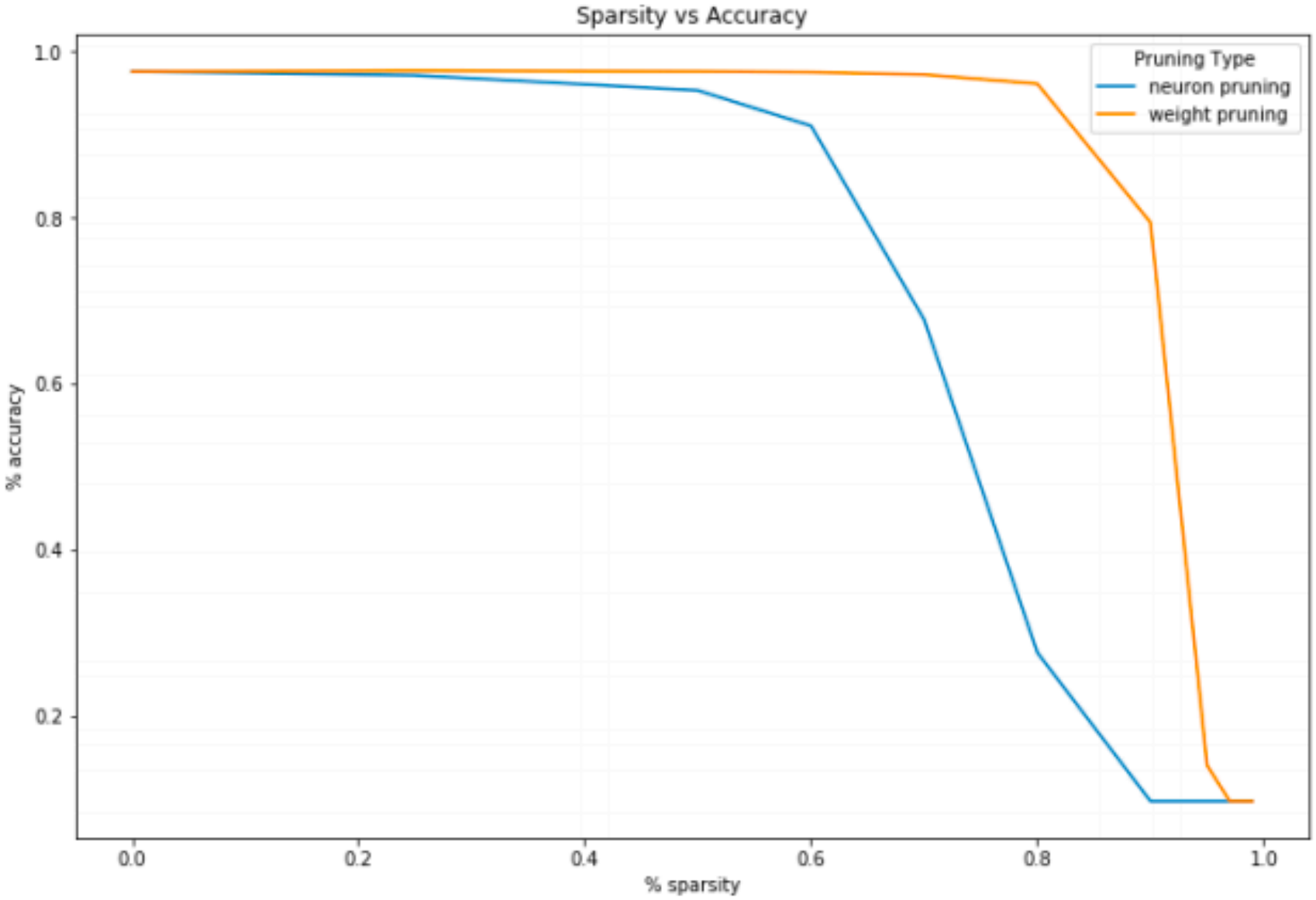
Iterative pruning



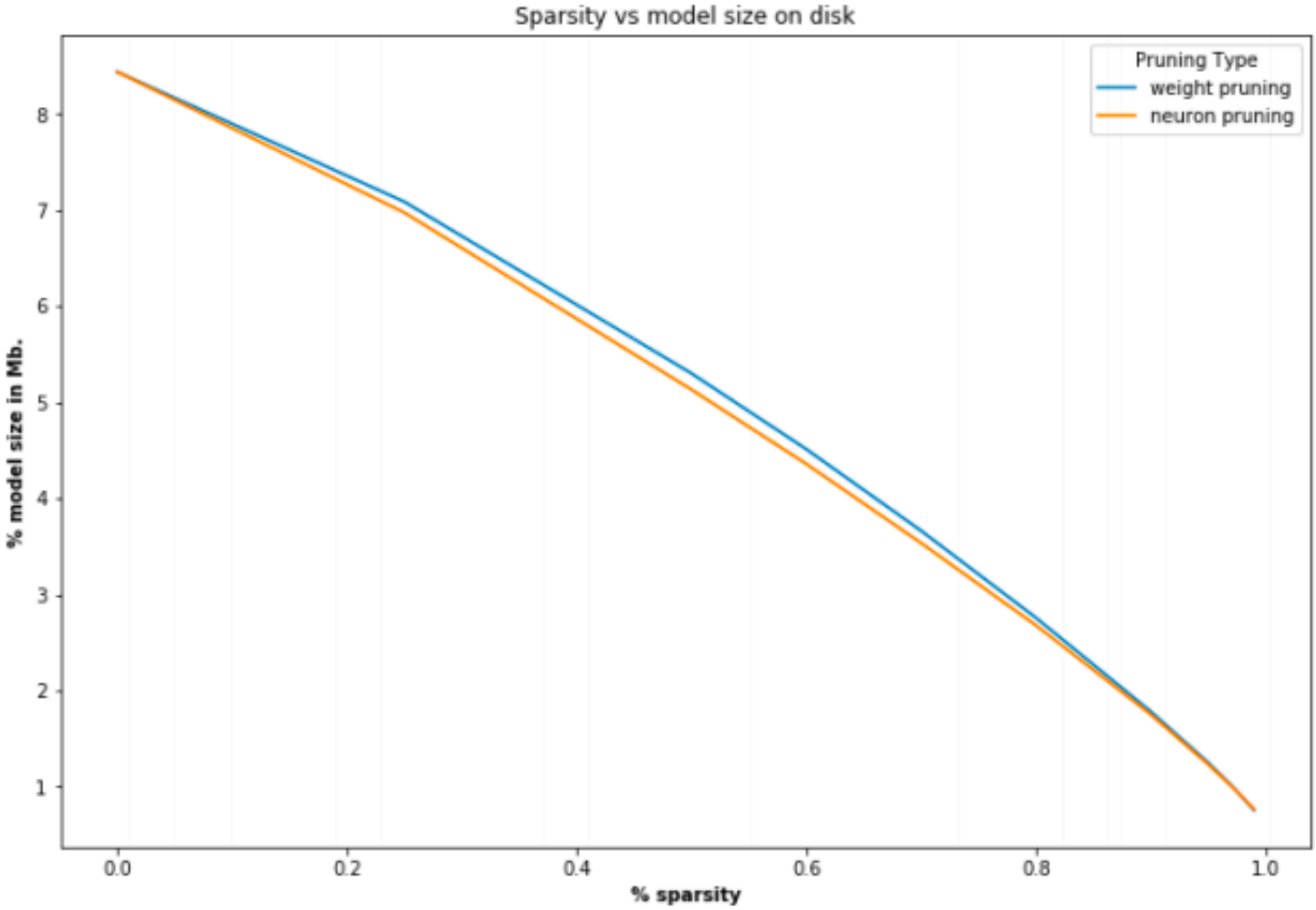
Если слишком сильно урезать нейронную сеть, можно ей навредить так, что она уже не восстановится.

Решение: итеративно повторять pruning.

Эффективность: точность



Эффективность: память



Quantization

- **Конвертируем веса и входящие значения из float в integer-8**
- **В 4 раза меньше памяти**
- **На некоторых устройствах ускоряем вычисления**

Предположим изначальные значения сети лежат в промежутке **[-a; a)**

Тогда наши значения преобразуются $x \mapsto \left\lfloor 128 \frac{x}{a} \right\rfloor$

Совершаем вычисления

После производим обратное преобразование $x \mapsto \frac{ax}{16384}$

$$\begin{pmatrix} -0.18120981 & -0.29043840 \\ 0.49722983 & 0.22141714 \end{pmatrix} \begin{pmatrix} 0.77412377 \\ 0.49299395 \end{pmatrix} = \begin{pmatrix} -0.28346319 \\ 0.49407474 \end{pmatrix}$$

$$\begin{pmatrix} -24 & -38 \\ 63 & 28 \end{pmatrix} \begin{pmatrix} 99 \\ 63 \end{pmatrix} = \begin{pmatrix} 4770 \\ 8001 \end{pmatrix} \longrightarrow \begin{pmatrix} -0.2911377 \\ 0.48834229 \end{pmatrix}$$

Quantization in practice

Два главных типа:

- **Post-training:** тренируем модель на Float-32, затем переводим веса в Integer-8. Преимущество: просто применять. Недостаток: потеря точности.
- **Quantization-aware training:** переводим веса в Integer-8 затем тренируем. Даже градиентный спуск выполняем для Int-8. Лучше итоговая точность, но сложнее в реализации.

Technique	Data requirements	Size reduction	Accuracy	Supported hardware
Post-training float16 quantization	No data	Up to 50%	Insignificant accuracy loss	CPU, GPU
Post-training dynamic range quantization	No data	Up to 75%	Accuracy loss	CPU, GPU (Android)
Post-training integer quantization	Unlabelled representative sample	Up to 75%	Smaller accuracy loss	CPU, GPU (Android), EdgeTPU, Hexagon DSP
Quantization-aware training	Labelled training data	Up to 75%	Smallest accuracy loss	CPU, GPU (Android), EdgeTPU, Hexagon DSP

Quantization efficiency

Model	Top-1 Accuracy (Original)	Top-1 Accuracy (Post Training Quantized)	Top-1 Accuracy (Quantization Aware Training)	Latency (Original) (ms)	Latency (Post Training Quantized) (ms)	Latency (Quantization Aware Training) (ms)	Size (Original) (MB)	Size (Optimized) (MB)
Mobilenet-v1-1-224	0.709	0.657	0.70	124	112	64	16.9	4.3
Mobilenet-v2-1-224	0.719	0.637	0.709	89	98	54	14	3.6
Inception_v3	0.78	0.772	0.775	1130	845	543	95.7	23.9
Resnet_v2_101	0.770	0.768	N/A	3973	2868	N/A	178.3	44.9

Matrix factorization: SVD

- Уменьшение памяти в плотных слоях в 5-13 раз
- Вычисления за $O(nmt + nt^2 + ntk)$ вместо $O(nmk)$

SVD: Compression Results

Trained on ImageNet 2012 database, then compressed

5 convolutional layers, 3 fully connected layers, softmax output layer

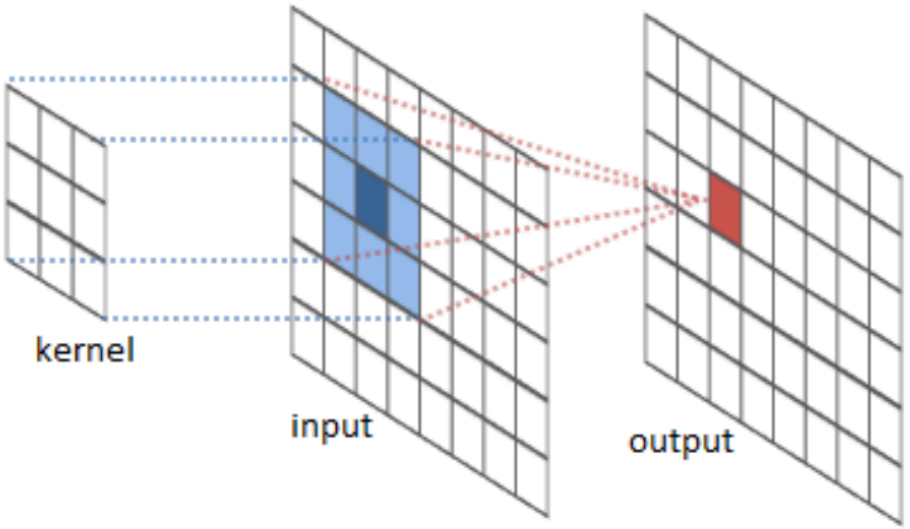
Approximation method	Number of parameters	Approximation hyperparameters	Reduction in weights	Increase in error
Standard FC	NM			
FC layer 1: Matrix SVD	$NK + KM$	$K = 250$ $K = 950$	$13.4\times$ $3.5\times$	0.8394% 0.09%
FC layer 2: Matrix SVD	$NK + KM$	$K = 350$ $K = 650$	$5.8\times$ $3.14\times$	0.19% 0.06%
FC layer 3: Matrix SVD	$NK + KM$	$K = 250$ $K = 850$	$8.1\times$ $2.4\times$	0.67% 0.02%

K refers to rank of approximation, t in the previous slides.

Matrix factorization: Flattened Convolutions

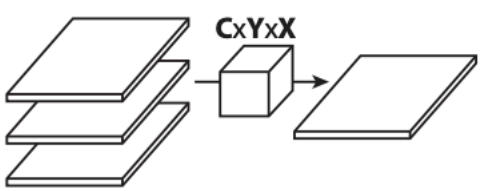
$$y[m,n] = x[m,n] * h[m,n] = \sum_{j=-\infty}^{\infty} \sum_{i=-\infty}^{\infty} x[i,j] \cdot h[m-i, n-j]$$

Самые большие
затраты времени тут:

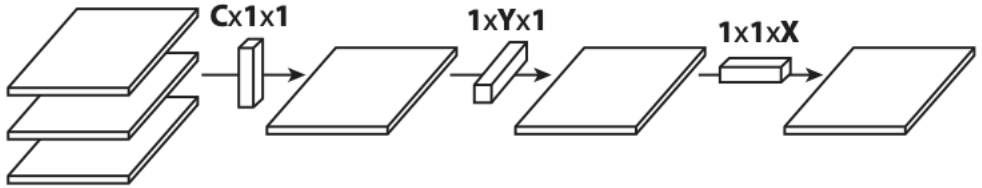


$$F(x,y) = I * W$$

Matrix factorization: Flattened Convolutions



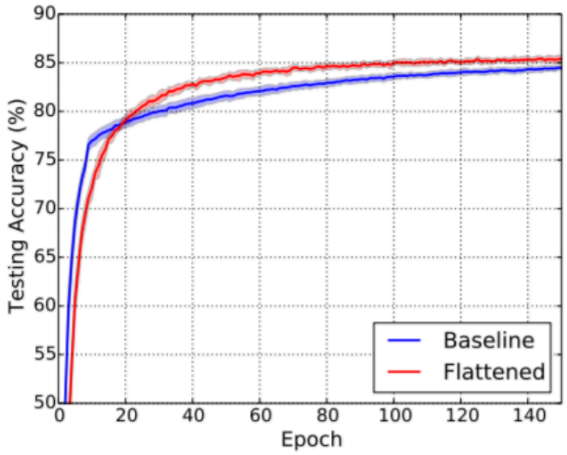
(a) 3D convolution

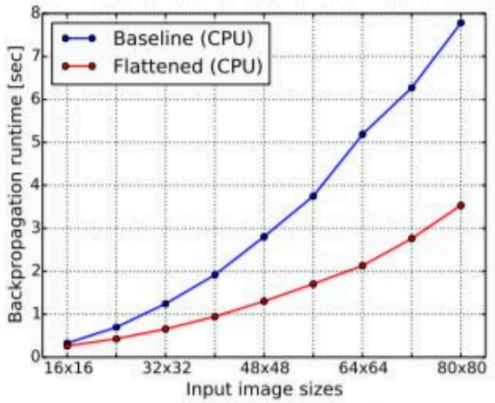


(b) 1D convolutions over different directions

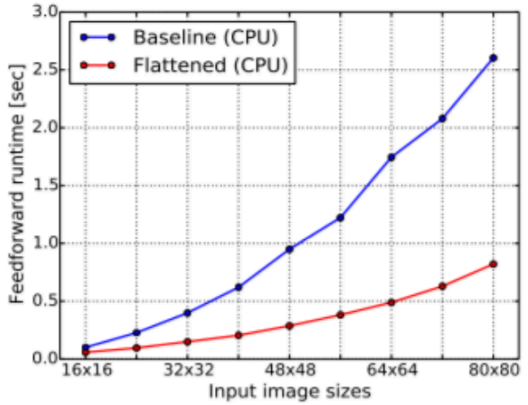
Compression and Speedup:

- Parameter reduction: $O(XYC)$ to $O(X+Y+C)$
- Operation reduction: $O(mnCXY)$ to $O\big(mn(C+X+Y)\big)$ (where $W_f \in \mathbb{R}^{m\times n}$)





(c) Backpropagation on CPU



(a) Feedforward on CPU

Matrix factorization: Tensor Train

- Одно из обобщений SVD
- Позволяет обучать слои, сжатые до 200000 раз без значительного увеличения ошибки
- Как и у обычной нейронной сети обучение с SGD

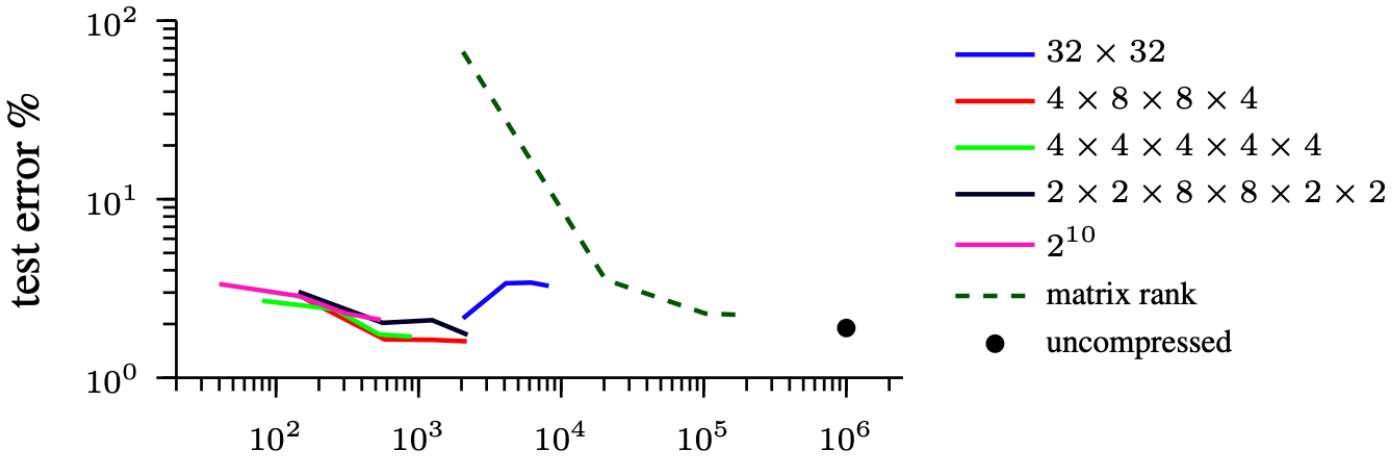
Operation	Time	Memory
FC forward pass	$O(MN)$	$O(MN)$
TT forward pass	$O(dr^2m\max\{M,N\})$	$O(r\max\{M,N\})$
FC backward pass	$O(MN)$	$O(MN)$
TT backward pass	$O(d^2\,\mathrm{r}^4\,m\max\{M,N\})$	$O(\mathrm{r}^3\max\{M,N\})$

$m_1 \times \ldots \times m_d$ — Входные ранги

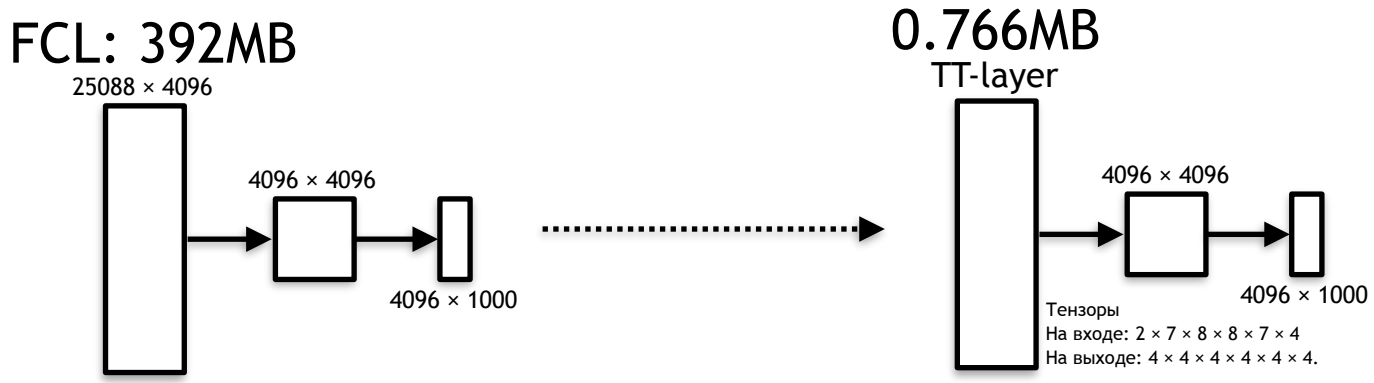
$n_1 \times \ldots \times n_d$ — Выходные ранги

$m = max(m1, ..., md)$

$r = TTrank_{max}$



Matrix factorization: Tensor Train



Type	1 im. time (ms)	100 im. time (ms)
CPU fully-connected layer	16.1	97.2
CPU TT-layer	1.2	94.7
GPU fully-connected layer	2.7	33
GPU TT-layer	1.9	12.9

Architecture	TT-layers compr.	vgg-16 compr.	vgg-19 compr.	vgg-16 top 1	vgg-16 top 5	vgg-19 top 1	vgg-19 top 5
FC FC FC	1	1	1	30.9	11.2	29.0	10.1
TT4 FC FC	50 972	3.9	3.5	31.2	11.2	29.8	10.4
TT2 FC FC	194 622	3.9	3.5	31.5	11.5	30.4	10.9
TT1 FC FC	713 614	3.9	3.5	33.3	12.8	31.9	11.8
TT4 TT4 FC	37 732	7.4	6	32.2	12.3	31.6	11.7

ИТОГ

Pruning: удаление нейронов дает улучшение в памяти и скорости, удаление синапсов дает улучшение в памяти и в скорости, если поддерживается вычисление на разреженных матрицах на аппаратном уровне.

Quantization: уменьшение памяти в 4 раза, все еще надо расширять до изначальных значений, чтобы производить вычисления. Возможно улучшение в скорости

Pruning и Quantization: не дают возможности обучать с нуля

Matrix factorization: имеет существенную экономию в памяти, может давать ускорение в вычислении, дает возможность обучать с нуля, значительно сложнее в реализации

Источники

- https://ekamperi.github.io/docs/dnn_compression.pdf
- https://www1.cmc.edu/pages/faculty/BHunter/papers/deep-negative-matrix.pdf
- https://towardsdatascience.com/pruning-neural-networks-1bb3ab5791f9#:~:text=Neural%20network%20pruning%20is%20a,removing%20unnecessary%20neurons%20or%20weights.
- https://www.hse.ru/data/2019/07/12/1477568723/%D0%93%D1%80%D0%B0%D1%87%D0%B5%D0%B2_%D1%80%D0%B5%D0%B7%D1%8E%D0%BC%D0%B5.pdf

Вопросы

- С какими проблемами можно столкнуться при обнулении весов в Pruning?
- Как и при каких условиях quantization дает улучшение в скорости работы и памяти?
- В чем преимущество матричного разложения над pruning и quantization?
- Преобразовать матрицу $\begin{pmatrix} 3 & -0.5 \\ 1 & 2 \end{pmatrix}$ методом quantization
- Совершить pruning над