Использование тензорных сетей для сжатия и восстановления изображений

Александр Моложавенко molozhavenko.aa@phystech.edu

Project Proposal

В этом проекте рассматривается возможность использования тензорных сетей для задач восстановления изображения по значениям в небольшом числе пикселей (tensor completion), а также для сжатия изображений.

1 Идея

С помощью тензорных сетей (*PEPS*, *Tensor Ring*, *Tensor Train*), алгоритма *Greedy-TN* [1], его модификаций и алгоритмами *HOSVD, ALS* найти оптимальные конфигурации для минимально возможного по количеству параметров представления изображений. Аналогично этими методами решается задача восста-новления изображений.

1.1 Проблема

Пусть имеется картинка $M \in \mathbb{R}^{3\times d_w \times d_h}$, задаваемая тремя матрицами пикселей (фильтр синего, фильтр красного и фильтр зеленого) или трехмерным тензором.Преобразуем трёхиндексную матрицу M в много индексную матрицу (в тензор высокого порядка) по основанию b, то есть:

$$M \in \mathbb{R}^{3\times d_w \times d_h} \mapsto \mathcal{T} \in \mathbb{R}^{b\times b\times \cdots \times b}.$$

Количество мод полученного тензора, равняется $N = \log_b(3 \cdot d_h \cdot d_w)$. Количество элементов в таком представлении изображение есть $3 \cdot d_w \cdot d_h = b^N$. Однако, если к полученному тензору высокого порядка применить *Tensor Train Decomposition* из N тензоров свертке с максимальном рангом тензорного поезда m, то придется хранить всего лишь около $N \cdot b \cdot m^2$ элементов. Что при правильной оптимизации ранга разлажения может дать существенное сжатие.

В терминах статьи [1] введем в рассмотрение тензорную сеть $\mathrm{TN}(\mathcal{G}^{(1)}, \mathcal{G}^{(2)}, \dots, \mathcal{G}^{(N)})$, которая в свертке по заданным в ней модам даёт тензор $\mathcal{W} \in \mathbb{R}^{b\times b\times \cdots \times b}$. В этих обозначениях проблемы восстановления и сжатия изображений перепишутся следующим образом:

1.1.1 Image compression

$$\|\mathcal{T} - \mathrm{TN}(\mathcal{G}^{(1)}, \mathcal{G}^{(2)}, \dots, \mathcal{G}^{(N)})\|_F^2 \to \min_{\mathcal{G}^{(1)}, \mathcal{G}^{(2)}, \dots, \mathcal{G}^{(N)}}, \tag{1}$$

причём в зависимости от выбранного метода оптимизации данной функции будут или не будут накладываться дополнительные условия связи между *core*-тензорами тензорной сети (адаптивный метод *Greedy-TN*, например автоматически подбирает нужные связи)

1.1.2 Image Completion

$$\frac{1}{|\Omega|} \sum_{(i_1,\dots,i_N)\in\Omega} \left(\mathcal{T}_{i_1,\dots,i_N} - \mathrm{TN}(\mathcal{G}^{(1)}, \mathcal{G}^{(2)}, \dots, \mathcal{G}^{(N)})_{i_1,\dots,i_N}\right)^2 \to \min_{\mathcal{G}^{(1)}, \mathcal{G}^{(2)}, \dots, \mathcal{G}^{(N)}}, \tag{2}$$

где Ω – множество индексов, значения под которыми нам известны (индексы известных частей картинки после её отображения в тензор высокого порядка).

2 Литературный обзор

2.1 Adaptive tensor learning with tensor networks

В работе идет речь про тензорные разложения (*TT, TR и Tucker*) и метод *Greedy-TN* поиска оптимального разложения тензора с помощью TensorNetwork для восстановления изображения.

2.1.1 TensorNetwork

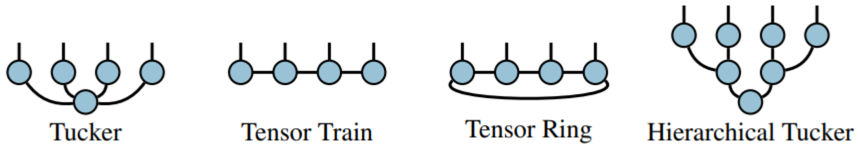

Figure 1: Тензорная сеть для тензорных разложений

Построенная тензорная сеть является решением задачи минимизации функции потерь $\mathcal{L}$. Рассмотрим пример: $\mathcal{T} \in \mathbb{R}^{d_1 \times d_2 \times d_3 \times d_4}$ - мы хотим его разложить в TT, тогда мы ищем:

$$\min_{\substack{\mathcal{G}^{(1)} \in \mathbb{R}^{d_1 \times r_1 \times 1 \times 1}; \mathcal{G}^{(2)} \in \mathbb{R}^{r_1 \times d_1 \times r_2 \times 1}; \\ \mathcal{G}^{(3)} \in \mathbb{R}^{1 \times r_2 \times d_3 \times r_3}; \mathcal{G}^{(4)} \in \mathbb{R}^{1 \times 1 \times r_3 \times d_4}}} \mathcal{L}(TN(\mathcal{G}^{(1)}, \mathcal{G}^{(2)}, \mathcal{G}^{(3)}, \mathcal{G}^{(4)})) \tag{3}$$

Но в общем случае тензорная сеть может иметь больше связей и выглядеть, например, вот так:

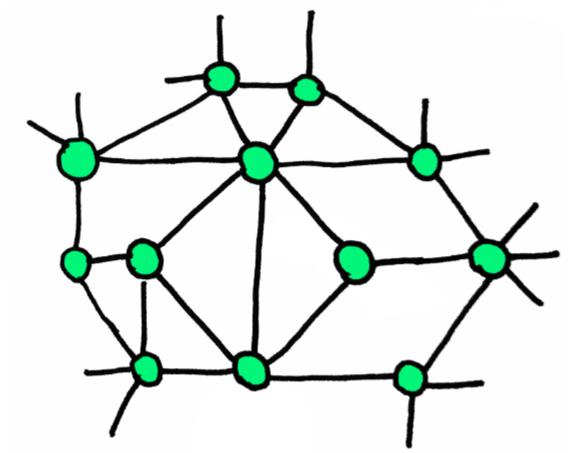

Figure 2: Вид тензорной сети

2.1.2 Greedy Algorithm

Из $\mathcal{T} \in \mathbb{R}^{d_1 \times d_2 \cdots \times d_p}$ строим его приближение core-тензорами $\mathcal{G}^{(i)} \in \mathbb{R}^{R_{1,i} \times \cdots \times R_{i-1,i} \times d_i \times R_{i,i+1} \times \cdots R_{i,p}}$.

Операции алгоритма описаны на 3

Algorithm 1 `Greedy-TN`: Greedy algorithm for tensor network structure learning.

Input: Loss function $\mathcal{L} : \mathbb{R}^{d_1 \times \cdots \times d_p} \to \mathbb{R}$, splitting node threshold ε.

// Initialize tensor network to a random rank one tensor and optimize loss function.
$R_{i,j} \leftarrow 1$ for $1 \leq i < j \leq p$
Initialize core tensors $\boldsymbol{\mathcal{G}}^{(i)} \in \mathbb{R}^{\times R_{1,i} \times \cdots \times R_{i-1,i} \times d_i \times R_{i,i+1} \times \cdots \times R_{i,p}}$ randomly
$(\boldsymbol{\mathcal{G}}^{(1)}, \cdots, \boldsymbol{\mathcal{G}}^{(p)}) \leftarrow \texttt{optimize}\ \mathcal{L}(\mathrm{TN}(\boldsymbol{\mathcal{G}}^{(1)}, \cdots, \boldsymbol{\mathcal{G}}^{(p)}))$ w.r.t. $\boldsymbol{\mathcal{G}}^{(1)}, \cdots, \boldsymbol{\mathcal{G}}^{(p)}$
repeat
    $(i, j) \leftarrow \texttt{find-best-edge}(\mathcal{L}, (\boldsymbol{\mathcal{G}}^{(1)}, \cdots, \boldsymbol{\mathcal{G}}^{(p)}))$
    // Weight transfer
    $\hat{\boldsymbol{\mathcal{G}}}^{(k)} \leftarrow \boldsymbol{\mathcal{G}}^{(k)}$ for $k \in [p] \setminus \{i, j\}$; $R_{i,j} \leftarrow R_{i,j} + 1$
    $\hat{\boldsymbol{\mathcal{G}}}^{(i)} \leftarrow \texttt{add-slice}(\boldsymbol{\mathcal{G}}^{(i)}, j)$ *// add new slice to the jth mode of* $\boldsymbol{\mathcal{G}}^{(i)}$
    $\hat{\boldsymbol{\mathcal{G}}}^{(j)} \leftarrow \texttt{add-slice}(\boldsymbol{\mathcal{G}}^{(j)}, i)$ *// add new slice to the ith mode of* $\boldsymbol{\mathcal{G}}^{(j)}$
    // Optimize new tensor network structure
    $(\boldsymbol{\mathcal{G}}^{(1)}, \cdots, \boldsymbol{\mathcal{G}}^{(p)}) \leftarrow \texttt{optimize}\ \mathcal{L}(\mathrm{TN}(\boldsymbol{\mathcal{G}}^{(1)}, \cdots, \boldsymbol{\mathcal{G}}^{(p)}))$ from init. $\hat{\boldsymbol{\mathcal{G}}}^{(1)}, \cdots, \hat{\boldsymbol{\mathcal{G}}}^{(p)}$
    // Add internal nodes if possible (number of cores p may be increased after this step)
    $(\boldsymbol{\mathcal{G}}^{(1)}, \cdots, \boldsymbol{\mathcal{G}}^{(p)}) \leftarrow \texttt{split-nodes}((\boldsymbol{\mathcal{G}}^{(1)}, \cdots, \boldsymbol{\mathcal{G}}^{(p)}), \varepsilon)$
until Stopping criterion

Figure 3: Greedy Algorithm

2.2 Matrix Product State / Tensor Train

Matrix Product State (*MPS*) или *tensor train* (*TT*) - это факторизация тензора с N индексами в цепочечное произведение трехиндексных тензоров. Это частный случай tree tensor network (*TTN*).

TT факторизация тензора T может быть представлена в графической нотации Пенроуза

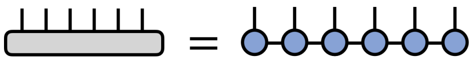

Figure 4: TT факторизация в нотации Пенроуза

В то же время, *TT* факторизация тензора T представима в традиционной нотации:

$$T^{s_1 s_2 s_3 s_4 s_5 s_6} = \sum_{\{\alpha\}} A^{s_1}_{\alpha_1} A^{s_2}_{\alpha_1 \alpha_2} A^{s_3}_{\alpha_2 \alpha_3} A^{s_4}_{\alpha_3 \alpha_4} A^{s_5}_{\alpha_4 \alpha_5} A^{s_6}_{\alpha_5} \tag{4}$$

где α_i - это размерности тензоров A.

Важно понимать, что тензоры в тензорном поезде соединяются с помощью ребер, имеющие размерности d_i. К примеру, если есть тензор $T^{s_1 s_2 \cdots s_N}$, имеющий N ребер с размерностями d, то тензор всегда может быть представлен в виде *TT* с размерностью $m = d^{\frac{N}{2}}$

К тому же тензор с N ребрами с размерностями d должен определяться d^N параметрами, но представление этого вектора с помощью *TT* с рангом m требует Ndm^2 параметров. При этом, кол-во параметров можно еще сильнее уменьшить.

Извлечение тензора размерности 1 из тензора стоит Nm^2. Если есть тензор

$$T^{s_1 s_2 s_3 \cdots s_N} = \sum_{\{\alpha\}} A^{s_1}_{\alpha_1} A^{s_2}_{\alpha_1 \alpha_2} A^{s_3}_{\alpha_2 \alpha_3} \cdots A^{s_N}_{\alpha_{N-1}}, \tag{5}$$

то извлечь из него тензор размерности 1 можно следующим образом – фиксируем s_j для каждого A. Затем свернем тензоры $A^{s_1}_{\alpha_1}$ и $A^{s_2}_{\alpha_1 \alpha_2}$ по α_1 и получим результирующий вектор L_{α_2}, который затем свернется с $A^{(s_3)}_{\alpha_2 \alpha_3}$. Продолжая то же действие, мы получим $s_1, s_2, s_3, \ldots, s_N$ компонент через последовательность векторно-матричных умножений.

Что касается произведения двух TT, то имея два тензора

$$T^{s_1 s_2 s_3 s_4 s_5 s_6} = \sum_{\{\alpha\}} A^{s_1}_{\alpha_1} A^{s_2}_{\alpha_1 \alpha_2} A^{s_3}_{\alpha_2 \alpha_3} A^{s_4}_{\alpha_3 \alpha_4} A^{s_5}_{\alpha_4 \alpha_5} A^{s_6}_{\alpha_5} \tag{6}$$

и

$$W^{s_1 s_2 s_3 s_4 s_5 s_6} = \sum_{\{\beta\}} B^{s_1}_{\beta_1} B^{s_2}_{\beta_1 \beta_2} B^{s_3}_{\beta_2 \beta_3} B^{s_4}_{\beta_3 \beta_4} B^{s_5}_{\beta_4 \beta_5} B^{s_6}_{\beta_5}, \tag{7}$$

мы получим

$$\langle T, W \rangle = \sum_{\{\mathbf{s}\}} T^{s_1 s_2 s_3 s_4 s_5 s_6} W^{s_1 s_2 s_3 s_4 s_5 s_6} \tag{8}$$

Сложность такого алгоритма составит $Nm^3 d$.

Особенно мощной операцией является сжатие тензорной сети в форму TT. Для конкретики предположим, что мы хотим сжать TT размерности M до размерности m, так чтобы новый TT был как можно ближе к исходному, в смысле евклидова расстояния. Процедура сжатия начинается со свертки двух копий входной сети TT по всем внешним индексам, кроме последнего внешнего индекса. Чтобы эффективно вычислить свертку двух копий TT, формируются промежуточные тензоры $E_1, E_2, \ldots$ как показано на иллюстрации.

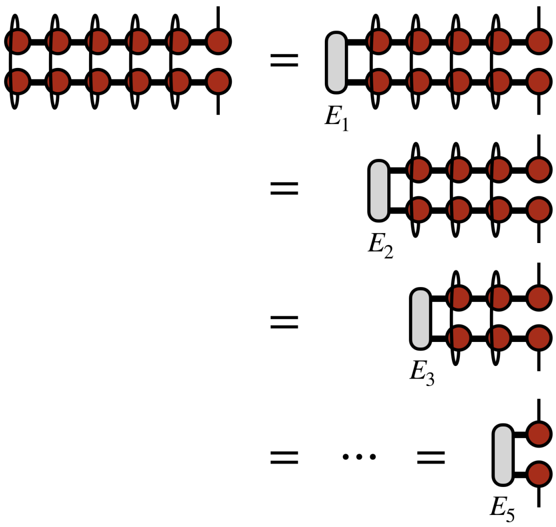

Figure 5: Преобразование свертки копий

Вычислив все E_j можно сформировать ρ_6 reduced density matrix - квадрат тензора, представленного сетью, суммируемый по всем, кроме его последнего индекса.

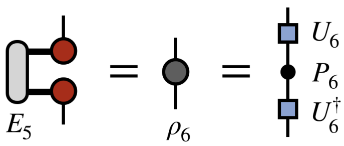

Figure 6: Reduced density matrix

Это эрмитова матрица, поэтому она всегда может быть диагонализирована унитарной U_6.

Чтобы сформировать новый тензор из сжатого TT, то мы формируем новую матрицу плотности и диагонализируем ее

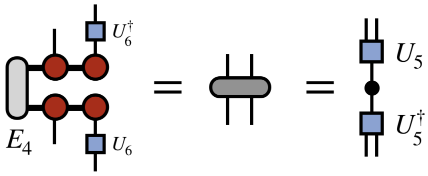

Figure 7: Следующий шаг

Получив U_5 как показано выше, вычисляется следующая матрица плотности, снова используя все предыдущие U тензоры для поворота и сжатия пространства, охватываемого всеми предыдущими внешними индексами

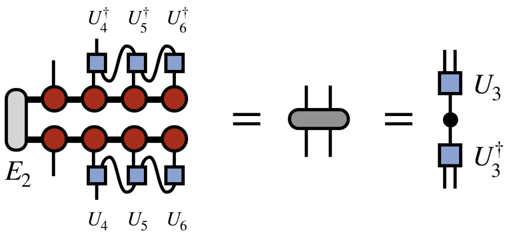

Figure 8: Следующая итерация

После того, как мы получаем все U тензоры, повторяя шаги выше, получим первый тензор нового TT с помощью следующей сверточной диаграммы

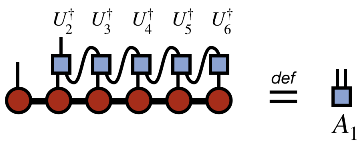

Figure 9: Сверточная диаграмма

Так соединив A_1 со всеми U_j получим новый сжатый тензор.

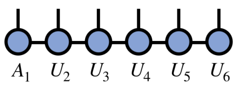

Figure 10: Сжатый тензор

2.3 Tensor Ring

В работе [2] исследуются свойства *Tensor Ring Decomposition*, которое заключается в представлении тензора высокого порядка в виде последовательности трёхмерных тензоров, которые умножаются «по кругу». Формально, пусть $\mathcal{T} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$ – тензор высокого порядка, а $\mathcal{Z}_k \in \mathbb{R}^{r_k \times n_k \times r_{k+1}}, k \in \overline{1, \ldots, d}$ – множество трёхмерных тензоров, такие что:

$$\mathcal{T}(i_1, i_2, \ldots, i_d) = \mathrm{Tr}\{(\mathcal{Z}_1(i_1)\mathcal{Z}_2(i_2) \cdots Z_d(i_d))\} = \mathrm{Tr}\left\{\prod_{k=1}^{d} \mathcal{Z}_k(i_k)\right\}, \tag{9}$$

где $\mathcal{T}(i_1, i_2, \ldots, i_d)$ – число в многоиндексной таблице $\mathcal{T}$ с индексами $i_1, i_2, \ldots, i_d$, а $\mathcal{Z}_k(i_k), \forall k \in \overline{1, \ldots, d}$ – срез трехмерного тензора $\mathcal{Z}_k$ с размерностью $r_k \times r_{k+1}$:

Tensor Ring Decomposition в отличие от *Tensor Train Decomposition* «устойчиво» по отношению к циклическим перестановкам рёбер тензора:

Теорема

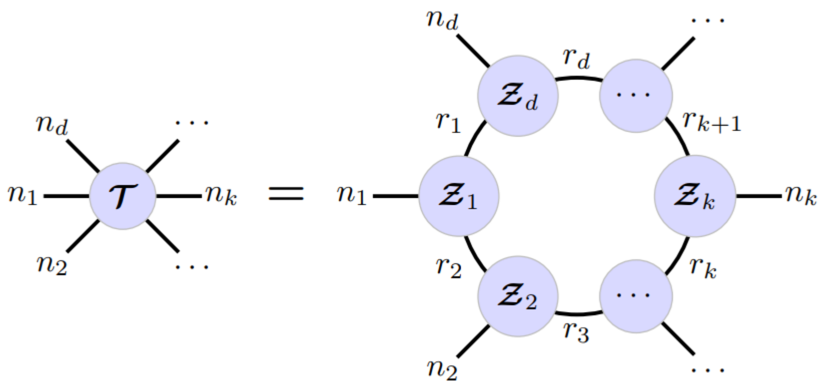

Figure 11: Графическое представление *Tensor Ring Decomposition*

Пусть $\mathcal{T} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$ – тензор высокого порядка с *Tensor Ring Decomposition* $\mathcal{R}(\mathcal{Z}_1, \mathcal{Z}_2, \ldots, \mathcal{Z}_d)$. Определим тензор $\vec{\mathcal{T}} \in \mathbb{R}^{n_{k+1} \times \cdots \times n_d \times n_1 \times \cdots \times n_k}$, полученный циклическим сдвигом рёбер исходного тензора $\mathcal{T}$ на k, тогда его *Tensor Ring Decomposition* будет $\mathcal{R}(\mathcal{Z}_{k+1}, \ldots, \mathcal{Z}_d, \mathcal{Z}_1, \ldots, \mathcal{Z}_k)$

Также статья даёт ответы на вопросы о некоторых алгебраических свойствах разложения *Tensor Ring* таких, как сложение и некоторые «хитрые» умножения.

2.4 Projected Entangled Pair States

Projected entangled pair states (PEPS) - обобщает тензорную сеть tensor train из одномерной сети в сеть на произвольном графе.

Тензорная диаграмма для $PEPS$ на конечной квадратной решетке имеет вид:

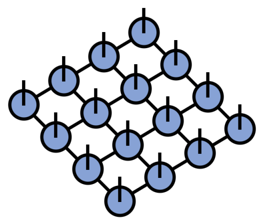

Figure 12: Тензорная диаграмма

Название PEPS происходит с той точки зрения на квантовую информацию, согласно которой общие тензорные сети можно рассматривать как максимально коррелированные тензоры в нескольких копиях тензорного индексного пространства, которые затем проецируются в одну копию индексного пространства.

3 Метрики качества

1) Относительное сжатие картинок для задачи сжатия
2) Относительная ошибка восстановления картинок для задачи восстановления
3) PSNR
4) Квадрат нормы Фробениуса разности тензоров

4 Примерный план

- Для задачи сжатия и восстановления: построить простую тензорную сеть для тензоризированной двумя различными методами картинки, используя *TT-SVD*, *ALS* и *Adam*.
- Проанализировать полученные результаты с помощью объявленных метрик.

- Сравнить работу алгоритма сжатия и восстановления с методами, работающими без тензоризации, например: SVD и скелетное разложение.

References

[1] Meraj Hashemizadeh, Michelle Liu, Jacob Miller, and Guillaume Rabusseau. Adaptive tensor learning with tensor networks. *CoRR*, abs/2008.05437, 2020.

[2] Qibin Zhao, Guoxu Zhou, Shengli Xie, Liqing Zhang, and Andrzej Cichocki. Tensor ring decomposition. *CoRR*, abs/1606.05535, 2016.