Matrix Decomposition for Adaptive Optimization Regularization

Lusine Airapetyan, Daniil Chesakov, Vsevolod Glazov, Evgeny Kovalev, Leonid Matyushin

Skolkovo Institute of Science and Technology, "Numerical Linear Algebra"

Moscow, 2018

AdaGrad algorithm background

Suppose that we have a smooth loss function $f : \mathbb{R}^n \to \mathbb{R}$, and the following minimization problem:

$$f(x) \to \min_{x \in \mathscr{X}}$$

Denote $g_k \equiv \nabla f_x(x_k)$ and $\mathbf{G}_k = [g_k \; g_{k-1} \dots \; g_1]$, where $\mathbf{G}_k \in \mathbb{R}^{n \times k}$.
In this notation the k-th step of optimization update:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \frac{\eta}{\sqrt{\mathbf{G}_k \mathbf{G}_k^T + \varepsilon \mathbf{I}}} \nabla f(\mathbf{x}_k),$$

Problem formulation

GGT uses the preconditioner from full-matrix AdaGrad.

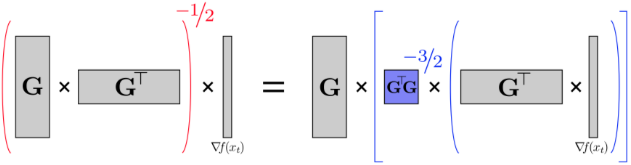
$$\mathbf{G}_k = [g_k g_{k-1} \ldots g_{k-r+1}], \quad \text{where } g_{k-t} = \beta_2^t \widetilde{\nabla} f\left(x_{k-t}\right), \text{ or } 0 \text{ if } t \geq k$$

where $\beta_2 \leq 1$ and $\widetilde{\nabla} f\left(x_{k-t}\right)$ is stochastic gradient.
GGT iterative step is:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \frac{\eta}{\sqrt{\mathbf{G}_k \mathbf{G}_k^T + \varepsilon \mathbf{I}}} \widetilde{\nabla} f(\mathbf{x}_k)$$

Key Idea

The inversion of the large low-rank matrix $\mathbf{G}\mathbf{G}^T \in \mathbb{R}^{n\times n}$ can be performed by diagonalizing the small matrix $\mathbf{G}^T\mathbf{G} \in \mathbb{R}^{r\times r}$.



Key Idea

$$\left[\left(\mathbf{G}\mathbf{G}^\top\right)^{1/2} + \varepsilon \mathbf{I}\right]^{-1} v = \frac{1}{\varepsilon} v + \mathbf{U}_r \left[\left(\boldsymbol{\Sigma}_r + \varepsilon \mathbf{I}_r\right)^{-1} - \frac{1}{\varepsilon}\mathbf{I}_r\right] \mathbf{U}_r^\top v \qquad (*)$$

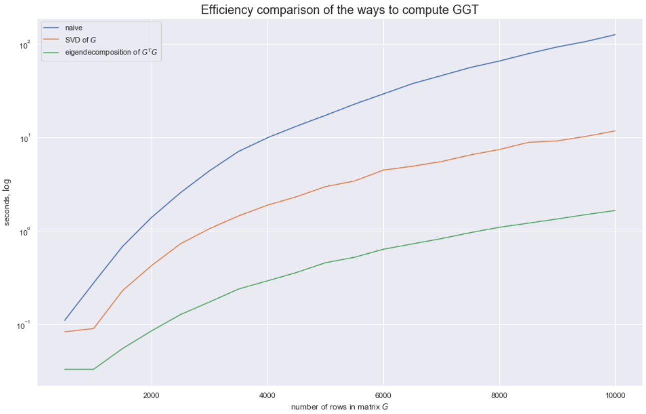
The first term is none other than an SGD update step. The rest can be computed by taking the eigendecomposition $\mathbf{G}^\top \mathbf{G} = \mathbf{V}\boldsymbol{\Sigma}_r^2 \mathbf{V}^\top$, giving $\mathbf{U}_r = \mathbf{G}\mathbf{V}\Sigma_r^{-1}$

Iterative step matrix computation

So, there are several ways to compute matrix $\left[\left(\mathbf{G}\mathbf{G}^\top\right)^{1/2} + \varepsilon\mathbf{I}\right]^{-1}$ which is used at the iterative step:

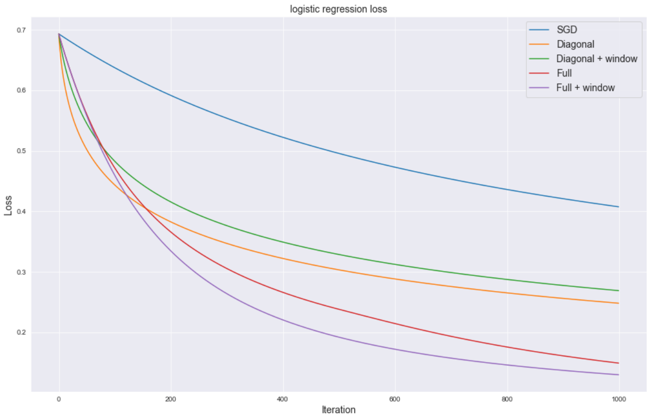
- naive way: use eigendecomposition of symmetric matrix $\mathbf{G}\mathbf{G}^\top$ to compute its square root, and then compute the inverse
- use $(*)$, obtain $\mathbf{U}_r$ and $\mathbf{\Sigma}_r$ via SVD decomposition of matrix $\mathbf{G}$
- use $(*)$, obtain $\mathbf{U}_r$ and $\mathbf{\Sigma}_r$ via eigendecomposition of matrix $\mathbf{G}^T\mathbf{G}$ as was described on the previous slide

Iterative step matrix computation



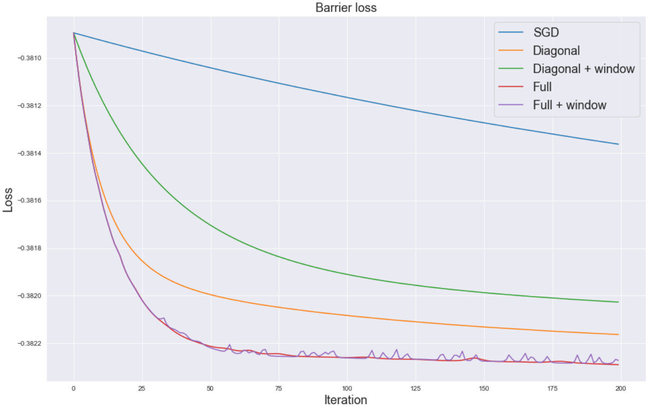
Results representation: syntetic data 1

We compared different full- and diagonal-matrix adaptive optimizers and SGD on the logistic regression problem on a set destibuted from an extremely anisotropic ($\sigma^2_{max}/\sigma^2_{min} \approx 10^4$) Gaussian distribution.



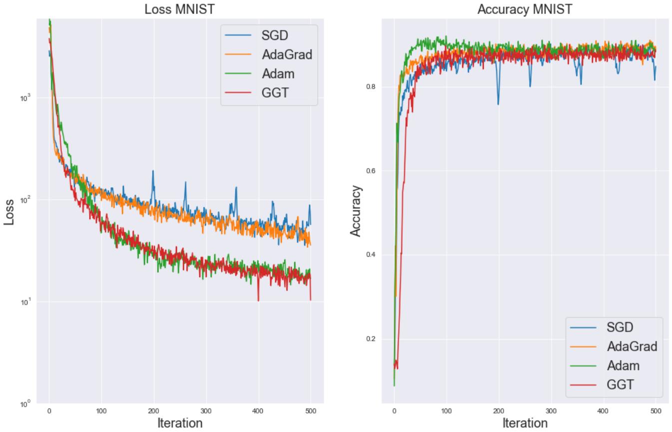
Results representation: syntetic data 2

We compared same optimizers on the the same set but now we minimized the barrier loss function: $f_i(w) = -\log\left(w^\top x_i + c_i\right)$ where c_i generated uniformly from $[0, 1]$.



Test on new data: MNIST

We compared modern state-of-the-art methods on a well known MNIST dataset. We DNN with two hidden fully connected layers with 256 nodes.



Our modifications

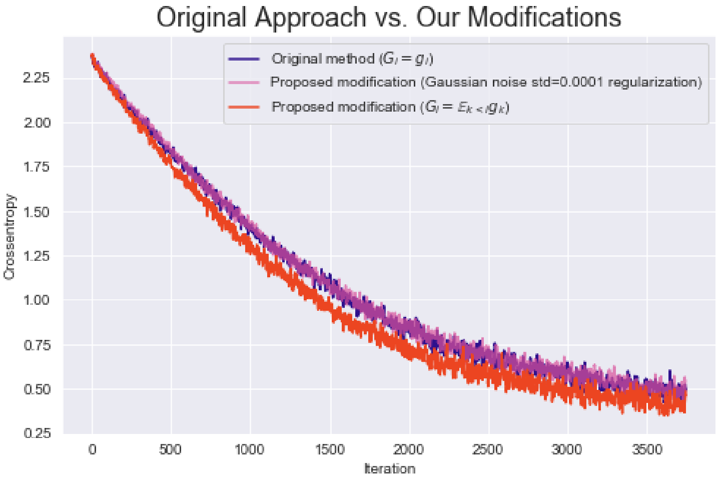
Original paper propose us to use the following matrix $\mathbf{G}_t$:

$$\mathbf{G}_t = \begin{pmatrix} g_t & g_{t-1} & \dots & g_{t-r+2} & g_{t-r+1} \end{pmatrix}$$

Where $g_{t-k} = \beta_2^k \nabla f(x_{t-k})$ (authors also suggest to use momentum with parameter $\beta_1 \approx 0.9$ and put $\beta_2 = 1$ on practice)
We considered several modifications of this method. The most important one is to replace matrix $\mathbf{G}_t$ by the following matrix:

$$\mathbf{G}_t = \begin{pmatrix} \frac{1}{r}\sum_{j=t-r+1}^{t} g_j & \frac{1}{r-1}\sum_{j=t-r+1}^{t-1} g_j & \dots & \frac{1}{2}\sum_{j=t-r+1}^{t-r+2} g_j & \sum_{j=t-r+1}^{t-r+1} g_j \end{pmatrix}$$

Our modifications



Low variance

High correlation (1.0)

Before

After

ROCs

Results

Model	Data	AUC-ROC
XGBoost	**tabular**	**0.9415**
Random Forest	tabular	0.9282
Logistic Regression	tabular	0.9003
KNN	tabular	0.8990
XGBoost	**TF – IDF**	**0.8622**
Random Forest	TF-IDF	0.8511
LSTM + Conv	**texts**	**0.8460**
KNN	TF-IDF	0.8350
Logistic Regression	TF-IDF	0.8316
LSTM	texts	0.8269

Feature importances

Conclusion

- Different approaches were compared
  - DL on texts
    - LSTM+Conv was better than LSTM
    - The worst results though
    - Probably model architecture should be more complex
  - ML on TF-IDF matrix
    - Best: XGBoost
  - ML on tabular data
    - Best: XGBoost
    - The best approach
- EDA was performed
  - Low variance, high correlation features were excluded
- Feature extraction from texts
  - Golden feature: stopwords share
- Model is applicable to a real-life scenario
  - It is interpretable, the quality is good
  - But better to train it on larger dataset