Optimizing NN with Kronecker-factored Approximate Curvature

Pavel Burnyshev
Optimization Class Project. MIPT

Introduction

Neural networks became the most popular area of machine learning. While stochastic gradient descent (SGD) with momentum works well enough in many situations, its performance declines dramatically as networks become deeper and more complex. The alternative is 2nd-order methods, for example natural gradient method.

$$\theta_{k+1} = \theta_k - \alpha_k F^{-1} \nabla h$$

Unfortunately, the cost of inverting the curvature matrix is prohibitive for neural networks, due to their high dimension. Kronecker-factored Approximate Curvature (K-FAC) (James Martens and Roger Grosse 2016). is an efficient method for approximating natural gradient descent in neural networks. K-FAC is based on an efficiently invertible approximation of a neural networks Fisher information matrix which is neither diagonal nor low-rank. It is derived by approximating various large blocks of the Fisher (corresponding to entire layers) as being the Kronecker product of two much smaller matrices. it is only several times more expensive to compute than the SGD, but the updates produced by K-FAC make much more progress optimizing the objective, which results in an algorithm that can be much faster than SGD with momentum.

Fisher Matrix

Neural network transforms its input x to an output $y = f(x, \theta)$ through a series of l layers, where θ is the vector of all network's parameters concatenated together

$$\theta = \left[\operatorname{vec}(W_1)^\top \operatorname{vec}(W_2)^\top \cdots \operatorname{vec}(W_\ell)^\top\right]^\top. \quad (1)$$

Thus our network defines a conditional model $P_{x|y}(\theta)$, with Fisher information matrix which can be viewed as an l by l block matrix

$$F = \mathbb{E}\left[\frac{\mathrm{d}\log p(y|x,\theta)}{\mathrm{d}\theta}\frac{\mathrm{d}\log p(y|x,\theta)^\top}{\mathrm{d}\theta}\right] = \mathbb{E}\left[\mathcal{D}\theta\mathcal{D}\theta^\top\right] = \begin{bmatrix} \mathbb{E}\left[\operatorname{vec}(\mathcal{D}W_1)\operatorname{vec}(\mathcal{D}W_1)^\top\right] & \cdots & \mathbb{E}\left[\operatorname{vec}(\mathcal{D}W_1)\operatorname{vec}(\mathcal{D}W_\ell)^\top\right] \\ \vdots & \ddots & \vdots \\ \mathbb{E}\left[\operatorname{vec}(\mathcal{D}W_\ell)\operatorname{vec}(\mathcal{D}W_1)^\top\right] & \cdots & \mathbb{E}\left[\operatorname{vec}(\mathcal{D}W_\ell)\operatorname{vec}(\mathcal{D}W_l)^\top\right] \end{bmatrix}. \quad (2)$$

Each block $F_{i,j}$ can be presented as:

$$F_{i,j} = \mathbb{E}\left[\operatorname{vec}(\mathcal{D}W_i)\operatorname{vec}\left(\mathcal{D}W_j\right)^\top\right] = \mathbb{E}\left[(\overline{a}_{i-1}\otimes g_i)\left(\overline{a}_{j-1}\otimes g_j\right)^\top\right] = \mathbb{E}\left[(\overline{a}_{i-1}\otimes g_i)\left(\overline{a}_{j-1}^\top \otimes g_j^\top\right)\right] = \mathbb{E}\left[\overline{a}_{i-1}\overline{a}_{j-1}^\top \otimes g_i g_j^\top\right], \quad (3)$$

where a_i is the input of the i layer, g_i is derivative of loss w.r.t. the inputs at layer i, $A \otimes \mathbf{B}$ denotes the Kronecker product between $A \in \mathbb{R}^{m\times n}$ and B:

$$A \otimes B = \begin{bmatrix} [A]_{1,1}B & \cdots & [A]_{1,n}B \\ \vdots & \ddots & \vdots \\ [A]_{m,1}B & \cdots & [A]_{m,n}B \end{bmatrix} \quad (4)$$

Initial approximation $\tilde{F}$ to F defines by the block-wise approximation:

$$F_{i,j} = \mathbb{E}\left[\overline{a}_{i-1}\overline{a}_{j-1}^\top \otimes g_i g_j^\top\right] \approx \mathbb{E}\left[\overline{a}_{i-1}\overline{a}_{j-1}^\top\right] \otimes \mathbb{E}\left[g_i g_j^\top\right] = \overline{A}_{i-1,j-1} \otimes G_{i,j} = \tilde{F}_{i,j}, \quad (5)$$

Additional approximations to $\tilde{F}$ and inverse computations

One of the methods to approximate $\tilde{F}^{-1}$ is the block-diagonal approximation. The analogue is using block-tridiagonal case. A natural choice for such an approximation $\breve{F}$ of $\tilde{F}$, is to take the block-diagonal of $\breve{F}$ to be that of $\tilde{F}$:

$$\breve{F} = \operatorname{diag}\left(\tilde{F}_{1,1}, \tilde{F}_{2,2}, \ldots, \tilde{F}_{\ell,\ell}\right) = \operatorname{diag}\left(\overline{A}_{0,0}\otimes G_{1,1}, \ldots, \overline{A}_{\ell-1,\ell-1}\otimes G_{\ell,\ell}\right)$$

Using the identity $(A\otimes B)^{-1} = A^{-1}\otimes B^{-1}$ we can compute the inverse of $\breve{F}$:

$$\breve{F}^{-1} = \operatorname{diag}\left(\overline{A}_{0,0}^{-1}\otimes G_{1,1}^{-1} \ldots, \overline{A}_{\ell-1,\ell-1}^{-1}\otimes G_{\ell,\ell}^{-1}\right) \quad (6)$$

In this way we need to compute the inverces of $2l$ smaller matrices instead of one large matrix inversion. To compute $u = \breve{F}v$ we can use identity $(A\otimes B)\operatorname{vec}(X) = \operatorname{vec}\left(BXA^\top\right)$ to get

$$U_i = G_{i,i}^{-1} V_i \overline{A}_{i-1,i-1}^{-1}, \quad (7)$$

u and v are the vectors of U_i and V_i respectively concatenated together.

Regularization techniques

Methods which use the exact Fisher work better with an adaptive Tikhonov regularization technique. It happens because K-FAC has no guarantee of being accurate up to 2nd-order.

For the block-diagonal approximation $\breve{F}$ of $\tilde{F}$ this means adding $(\lambda+\eta)I$ to each of the diagonal, blocks, which gives us new equation for each block:

$$\overline{A}_{i-1,i-1}\otimes G_{i,i} + (\lambda+\eta)I = \overline{A}_{i-1,i-1}\otimes G_{i,i} + (\lambda+\eta)I \otimes I.$$

For inverting that equation we can use next formula:

$$(A\otimes B \pm C\otimes D)^{-1} = (K_1\otimes K_2)(I\otimes I \pm S_1\otimes S_2)^{-1}\left(K_1^\top \otimes K_2^\top\right), \quad (8)$$

where $K_1 = A^{-1/2}E_1$ and $K_2 = B^{-1/2}E_2$.

Algorithm

Initialize θ_1

Choose λ, η

while θ_k **is not satisfactory do**

1. Choose $mini - batch\ size$ m.
2. Perform forward and backward pass to estimate the gradient $\nabla h(\theta_k)$.
3. Update the $\overline{A}_{i,j}$ and $\overline{G}_{i,j}$ using a_i, computed in forward pass.
4. If necessary recompute the approximate Fisher and compute Fisher inverse.
5. Compute the update δ.
6. $\theta_{k+1} := \theta_k + \delta$.
7. $k := k+1$.

Computational Costs

Following the author's most heuristics we can get next costs for number of operations:

K-FAC with block-diagonal approx.	$C_1\ell d^2 m + C_2\ell d^3$
K-FAC with block-threediagonal approx.	$C_1\ell d^2 m + C_4\ell d^3$
SGD	$C_1\ell d^2 m$

where l is the number of layers, d is the typical number of units in each layer, m is the mini-batch size.

Experiment

To investigate the applicability of K-FAC for optimization neural networks we consider classification problem using AlexNet with "Fashion-MNIST" dataset. The loss, accuracy and time of K-FAC was compared with SGD with momentum and Adam.

Results

To achieve the same results of loss and accuracy as K-FAC SGD and Adam spends several times more iterations. On the other hand, the overall time of K-FAC work is 5 times bigger.

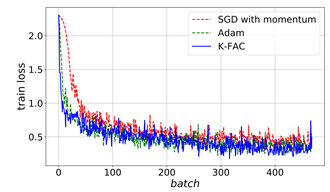

Figure 1: Train loss

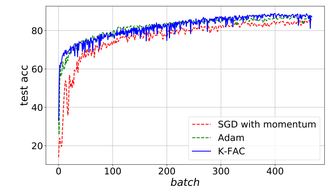

Figure 2: Accuracy of model

References

- Optimizing Neural Networks with Kronecker-factored Approximate Curvature
- New insights and perspectives on the natural gradient method
- Pytorch K-FAC implementation

Attachments

- Colab notebook