Optimizing Neural Networks with Kronecker-Factored Approximate Curvature

Josh Romoff & Riashat Islam

IFT6085 Paper Presentation

Reasoning and Learning Lab

28th February 2018

Training Neural Networks

While deep networks have rich modelling capacity, the complex dependency between parameters can make learning difficult

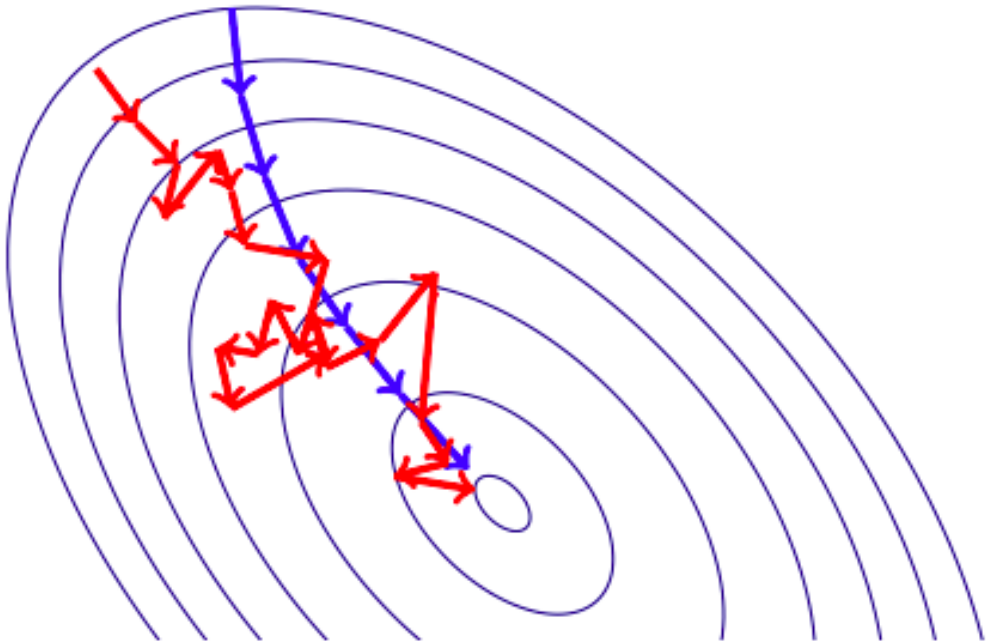
Training Neural Networks

This calls for better optimization techniques for training deep nets

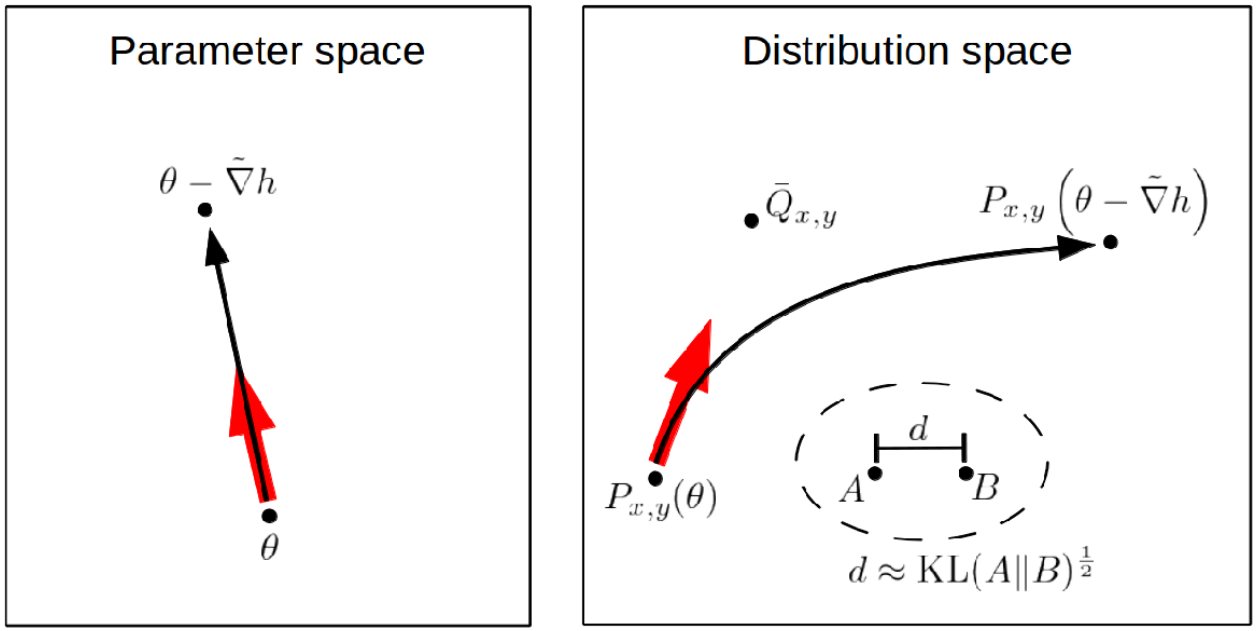
Overview

- Natural Gradients as alternative to SGD
- Fisher Information Matrix
- Approximations - KFAC
- Experimental Results
- Discussion

Stochastic Gradient Descent



Natural Gradient Descent



The Natural Gradient

$$\arg\min_{\Delta\theta} \mathcal{L}(\theta + \Delta\theta)$$
$$\text{s. t. } \mathbb{E}_{\mathbf{x}\sim\tilde{q}(\mathbf{x})}\left[KL(p_\theta(\mathbf{t}|\mathbf{x})||p_{\theta+\Delta\theta}(\mathbf{t}|\mathbf{x}))\right] = const.$$

- We use the expected value of the KL divergence between $p_\theta(t|x)$ and $p_{\theta+\delta\theta}(t|x)$ as a measure of the functional behaviour of $p_\theta(t|x)$ for different values of x

The Natural Gradient

Intuitively...

- Relies on the KL divergence between iterates
- Want to guarantee that the distribution of the new network will be similar to the old one
- Add a KL divergence constraint to the update, to ensure the new network will behave relatively similar to the old one

Why Natural Gradients

The KL divergence constraint ensure that we move along the functional manifold with constant speed, without being slowed by the curvature. It is a measure of how the probability density function changes, regardless of how it is parameterized.

- In mini-batch updates, some outliers may make drastic changes to network's parameters
- Stable learning process
- Natural gradients ensure monotonic parameter improvements

Learning locally robust to re-parameterizations of the model $\rightarrow$ *the functional behaviour of p does not depend on the parameterization of the model*

Why Natural Gradients

The gradients are of p_θ which acts as a proxy for the cost L. The KL constraint on the probability distributions ensures

- Each gradient update will make ϵ change to the model
- The model does not change by more than ϵ

This can provide some kind of robustness to overfitting.

- The model is not allowed to move too far in some direction, if moving along that direction changes the density computed by the model substantially.

Fisher Information Matrix

The metric for natural gradients is determined by the Fisher Information Matrix

$$F = \mathrm{E}\left[\frac{\mathrm{d}\log p(y|x,\theta)}{\mathrm{d}\theta}\frac{\mathrm{d}\log p(y|x,\theta)}{\mathrm{d}\theta}^{\top}\right] = \mathrm{E}[\mathcal{D}\theta\mathcal{D}\theta^{\top}]$$

ie, the covariance of the gradients of the model log probabilities w.r.t its paramters

The natural gradient is the direction obtained by

$$\nabla_N = F_\theta^{-1}\nabla \tag{1}$$

Difficulties with Natural Gradient

$$\nabla_N L(\theta) = \nabla L(\theta) F^{-1} \tag{2}$$

where F is the inverse of the Fisher information matrix.

- For large networks with millions of parameters, computing the inverse F^{-1} is computationally impractical.

We resort to approximation methods for efficiently computing the Fisher matrix F^{-1} and the natural gradient

- A key ingredient to designing optimization algorithms based on the natural gradients

Here, we will introduce the **KFAC** approximation method for efficiently computing the inverse of Fisher Information Matrix.

Kronecker Factored Fisher Approximation

We first note that the Fisher Matrix can be viewed as an ℓ by ℓ block matrix, where ℓ is the number of layers.

$$
\begin{aligned}
F &= \mathrm{E}\left[\mathcal{D}\theta\mathcal{D}\theta^\top\right] \\
&= \mathrm{E}\left[\left[\mathrm{vec}(\mathcal{D}W_1)^\top \ \mathrm{vec}(\mathcal{D}W_2)^\top \ \cdots \ \mathrm{vec}(\mathcal{D}W_\ell)^\top\right]^\top \left[\mathrm{vec}(\mathcal{D}W_1)^\top \ \mathrm{vec}(\mathcal{D}W_2)^\top \ \cdots \ \mathrm{vec}(\mathcal{D}W_\ell)^\top\right]\right] \\
&= \begin{bmatrix}
\mathrm{E}\left[\mathrm{vec}(\mathcal{D}W_1)\,\mathrm{vec}(\mathcal{D}W_1)^\top\right] & \mathrm{E}\left[\mathrm{vec}(\mathcal{D}W_1)\,\mathrm{vec}(\mathcal{D}W_2)^\top\right] & \cdots & \mathrm{E}\left[\mathrm{vec}(\mathcal{D}W_1)\,\mathrm{vec}(\mathcal{D}W_\ell)^\top\right] \\
\mathrm{E}\left[\mathrm{vec}(\mathcal{D}W_2)\,\mathrm{vec}(\mathcal{D}W_1)^\top\right] & \mathrm{E}\left[\mathrm{vec}(\mathcal{D}W_2)\,\mathrm{vec}(\mathcal{D}W_2)^\top\right] & \cdots & \mathrm{E}\left[\mathrm{vec}(\mathcal{D}W_2)\,\mathrm{vec}(\mathcal{D}W_\ell)^\top\right] \\
\vdots & \vdots & \ddots & \vdots \\
\mathrm{E}\left[\mathrm{vec}(\mathcal{D}W_\ell)\,\mathrm{vec}(\mathcal{D}W_1)^\top\right] & \mathrm{E}\left[\mathrm{vec}(\mathcal{D}W_\ell)\,\mathrm{vec}(\mathcal{D}W_2)^\top\right] & \cdots & \mathrm{E}\left[\mathrm{vec}(\mathcal{D}W_\ell)\,\mathrm{vec}(\mathcal{D}W_\ell)^\top\right]
\end{bmatrix}
\end{aligned}
$$

Kronecker Factored Fisher Approximation

Looking at one of the blocks, $F_{i,j}$, and by doing some re-arranging we get:

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

where we recall that $g_i = \mathcal{D}s_i$, $a_i = \phi_i(s_i)$ and $s_i = W_i a_{i-1}$.
So, each block can be viewed as the expected Kroenecker product between $\bar{A}_{i,j} = \bar{a}_{i-1}\bar{a}_{j-1}^T$ and $\bar{G}_{i,j} = \bar{g}_i\bar{g}_j^T$

Kronecker Factored Fisher Approximation

Now for the main approximation. Consider again one of the blocks, $F_{i,j}$, we assume the following:

$$F_{i,j} = \mathrm{E}\left[\bar{a}_{i-1}\bar{a}_{j-1}^{\top} \otimes g_i g_j^{\top}\right] \approx \mathrm{E}\left[\bar{a}_{i-1}\bar{a}_{j-1}^{\top}\right] \otimes \mathrm{E}\left[g_i g_j^{\top}\right] = \bar{A}_{i-1,j-1} \otimes G_{i,j} = \tilde{F}_{i,j}$$

So, we assume that the expected Kroenecker product between two matrices is equal to the Kroenecker product of the expectations; i.e that they are independent.

Further Approximations for Efficient Inversion

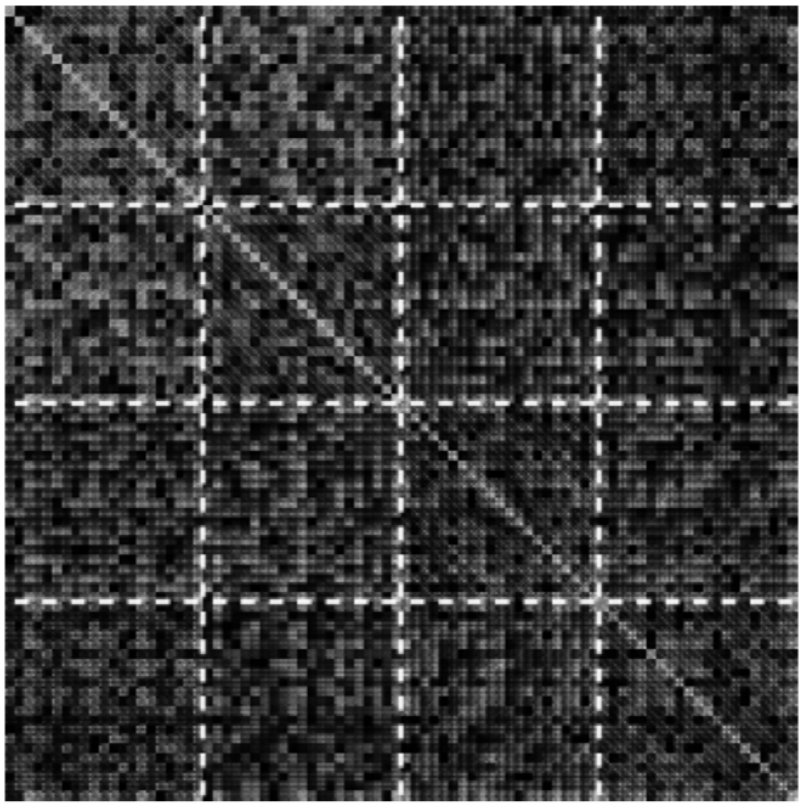
One can trade-off precision for computational savings (regarding the matrix inverse) by doing one of the following:

- Approximating $\tilde{F}^{-1}$ as block-diagonal
- Approximating $\tilde{F}^{-1}$ as block-tridiagonal

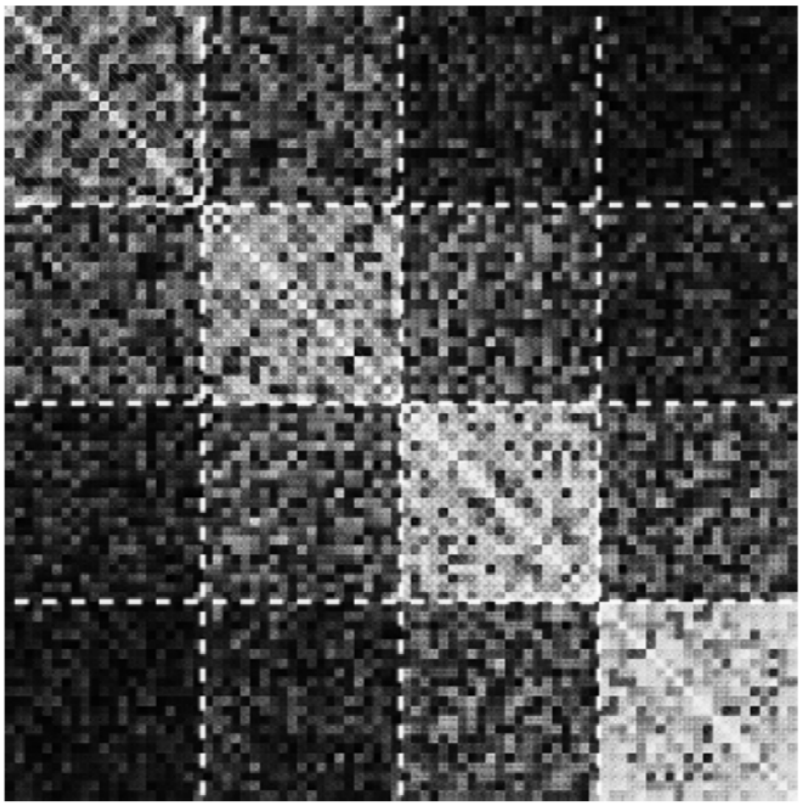
Intuitively, the block-diagonal approximation will be less precise but at substantial computational savings.

Experimental Results (Approximating $\tilde{F}^{-1}$)

$\tilde{F}$



$\tilde{F}^{-1}$

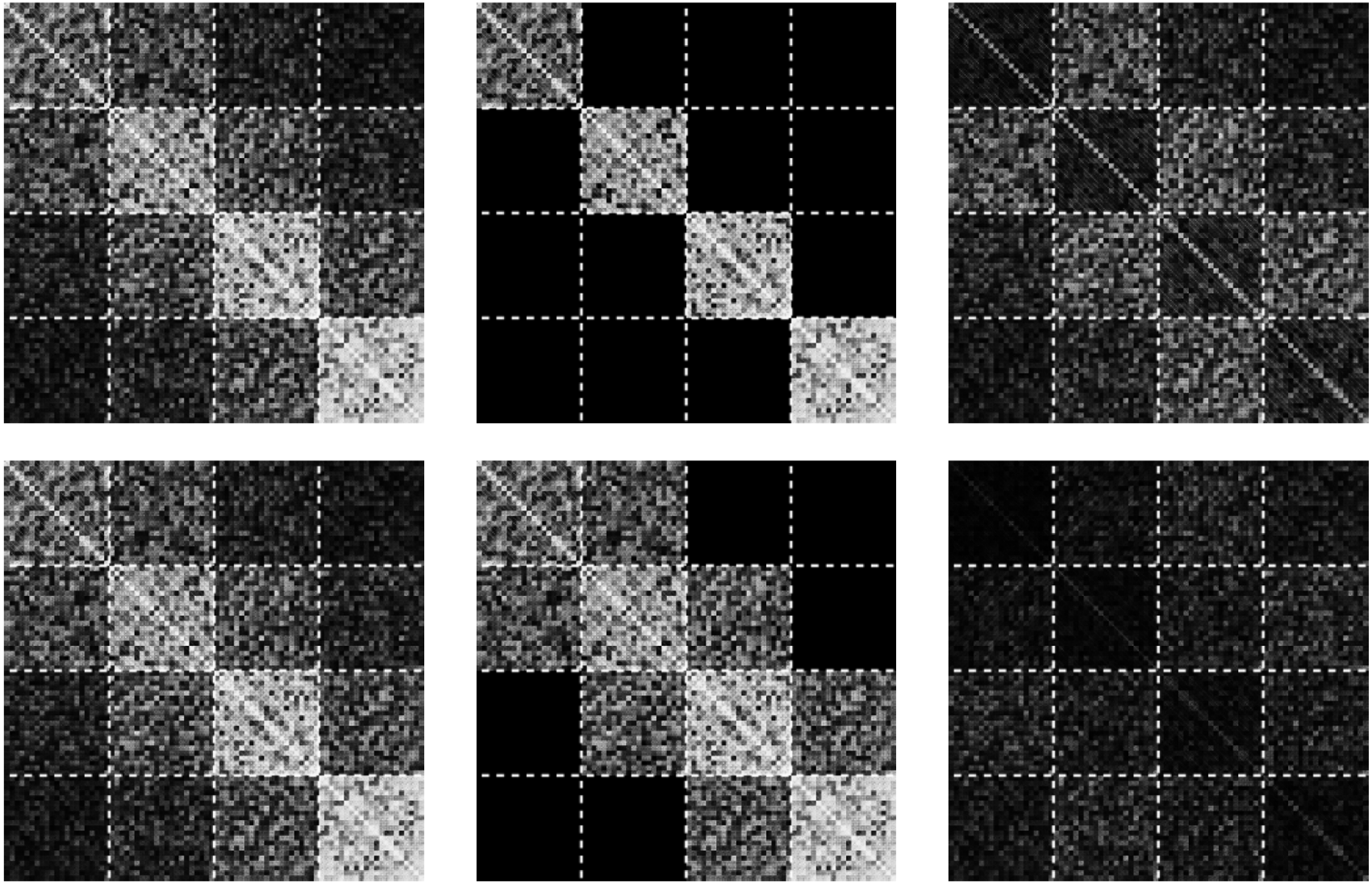


Experimental Results (Approximating $\tilde{F}^{-1}$)

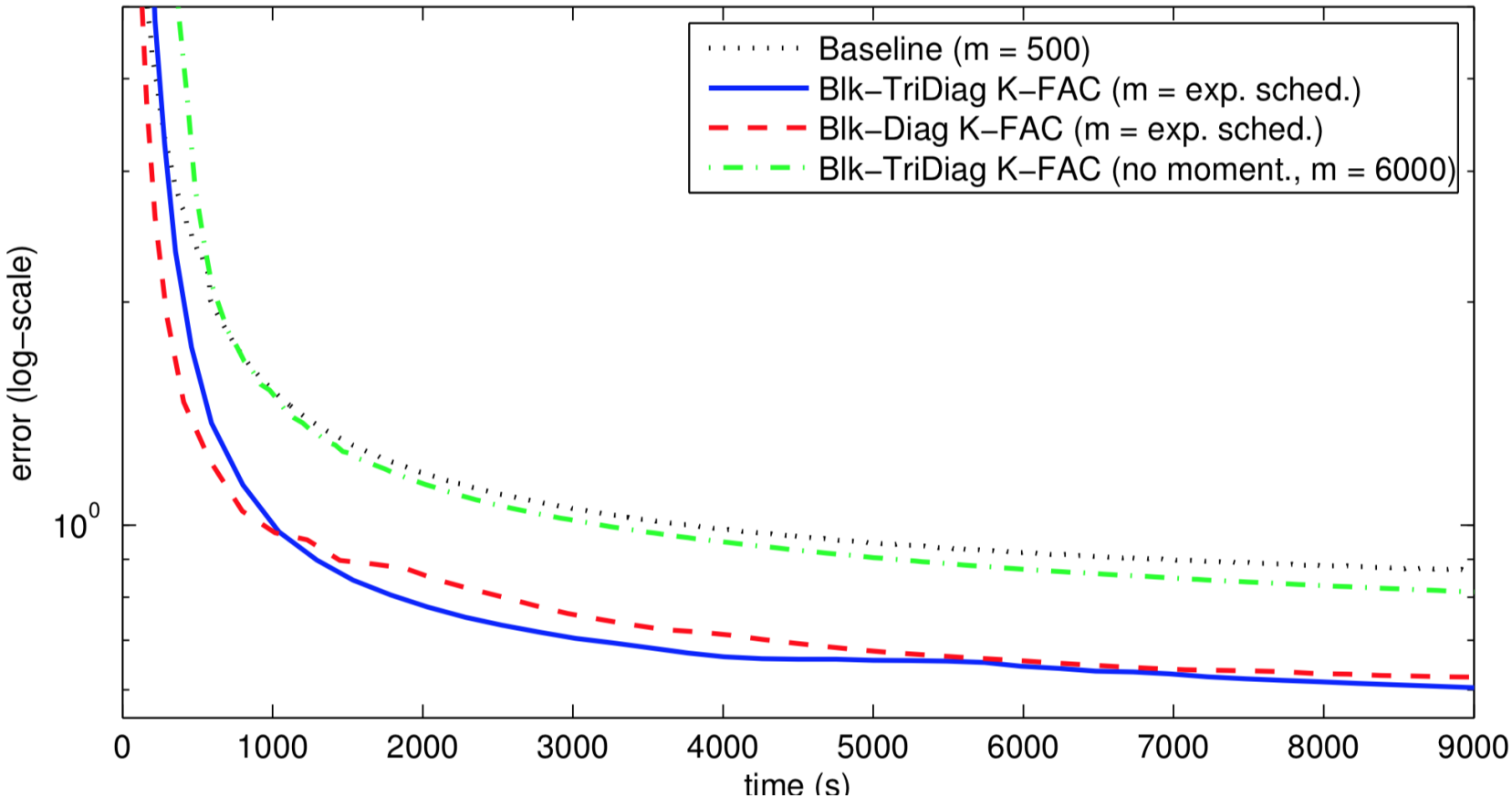
$\tilde{F}^{-1}$

$Diag$, $Triag$

$|\tilde{F}^{-1} - Diag|$, $|\tilde{F}^{-1} - Triag|$



Experimental Results (MNIST)



Summary

- Natural gradients as optimization techniques might be better
- But difficulty with the Fisher Information Matrix
- Need ways to approximate the Fisher matrix
- KFAC approximations to the rescue!

Thank You

The only stupid question is the one you were afraid to ask but never did
- Rich Sutton