Meta-Learning with Hessian Free Approach in Deep Neural Nets Training

Boyu Chen  
Fudan University  
17110180037@fudan.edu.cn

Wenlian Lu  
Fudan University  
wenlian@fudan.edu.cn

Abstract

Meta-learning is a promising method to achieve efficient training method towards deep neural net and has been attracting increase interests in recent years. But most of the current methods are still not capable to train complex neuron net model with long-time training process. In this paper, a novel second-order meta-optimizer, named Meta-learning with Hessian-Free (MLHF) approach, is proposed based on the Hessian Free approach as the framework. Two recurrent neural networks are established to generate the damping and the precondition matrix of this Hessian free framework. A series of techniques to meta-train the MLHF towards stable and reinforce the meta-training of this optimizer, including the gradient calculation of $H$, and use experiment replay on $w^0$. Numerical experiments on deep convolution neural nets, including CUDA-convnet and resnet18(v2), with datasets of cifar10 and ILSVRC2012, indicate that the MLHF shows good and continuous training performance during the whole long-time training process, i.e., both the rapid-decreasing early stage and the steadily-decreasing later stage, and so is a promising meta-learning framework towards elevating the training efficiency in real-world deep neural nets.

1 Introduction

The meta-learning in optimizing neural network, usually named as the learning-to-learn, has been attracting increasingly interests of the researchers of deep learning in the last few years [3, 8, 38, 21, 22, 30, 56, 10]. In comparison to the hand-crafted optimizers, for instance Stochastic Gradient Descent (SGD) as well as its variants, including ADAM [17], RMSprop [55], the methodology of meta-learning is to employ trained meta-optimizer, usually via recurrent neural networks (RNN), to infer descent directions, used to train the underlying neural networks, towards better learning performance. This methodology is promising, because it has been widely believed that neural network can “learn” a “more effective” descent direction than the existing ones.

A meta-learning method is generally twofold. One is a well-defined neural network that outputs the “learned” descent direction and can be heuristic, and a decomposition mechanism, also known as framework, to largely reduce the number of meta-parameters of the meta-optimizer and enhances its generality, i.e., the trained meta-optimizer can work for at least a type of neural net learning tasks. The major frameworks in the latest few years include coordinatewise framework [3] and hierarchical framework [38] via RNN. However, most of the current meta-learning methods can only work for simple back-propagation (BP) model with short-time training process, because they all shows unstable when training a large scale of deep neural net [38]. Hence, developing an efficient meta-optimizer and a good framework that is stable with acceptable computing cost, is still a challenge towards utilization of meta-learning to practical deep networks.

In this paper, we propose a novel second-order meta-optimizer, which utilizes the Hessian-Free method [23] as the framework. Specifically, the contribution and novelty of this paper include:
We realize the well-known the Hessian-Free method in meta-learning;
We improve the learning-to-learn losses of the recurrent neural networks of the meta-optimizer and utilize the experimental relay process in the meta-training;
This meta-optimizer shows the continuance of making effective progress for the long-time training process of practical deep neural networks, including CUDA-Convnet\[18\] and ResNet18(v2)\[14\].

Related Works

Meta-learning has a long history as long as the development of neural net itself. The early exploration was done by Schmidhuber \[33\] in 1980’s. Afterwards, based on this idea, a lot of works appeared to proposed diverse learning algorithms, for instance \[34\], \[29\], \[16\]. At the same time, Bengio et al. \[7\], \[5\], \[6\] introduced learning locally parameterized rules instead of back-propagation. In the very recent years, the framework of coordinatewise RNN proposed by Andrychowicz et al. \[3\] illuminated a promising orient towards a meta-learned optimizer can be employed to diverse neural network architectures.

The power of the framework of coordinatewise RNN inspired the development of meta-learning. Andrychowicz et al. \[3\] also employed Broyden–Fletcher–Goldfarb–Shanno algorithm (BFGS)\[4\] with the reverse of Hessian matrix regarded as the the memory, and coordinatewise RNN as the controller of an Neural Turing Machine\[12\]. However, storage of the reverse of Hessian matrix requires \(O(n^2)\) memory which is impossible to in a large-scale neural net. Li and Malik \[21\] proposed an similar approach at the same time, but the training algorithm of RNN in meta-optimizer realized by reinforcement learning. Ravi and Larochelle \[30\] profiled the method of \[3\] to few-shot learning tasks by using test error to train the meta-optimizer and applied for few-shot tasks. Chen et al. \[8\] utilized RNN to output the queue point of Bayesian optimization to train the neural net, instead of outputting descent directions. Finn et al. \[10\] proposed the Model-Agnostic Meta-Learning method by proposing a new parameter initialization strategy to enhance generalization of meta-learning method.

In contrast, Wichrowska et al. \[38\] addressed the problems of stabilization and generalization of \[3\]. They proposed a hierarchical architecture framework than the coordinatwise framework. For the first time, they proposed the method of learning-to-learn that has been applied to train large-scale deep neural nets like Inception v3 and ResNet v2 on ILSVRC2012 with big datasets. However, the performance is not very ideal.

2 Preliminaries

We consider a neural net formulated by \(z = f(x, w)\), where \(x\) stands for the input training data, \(w\) for all parameters and \(z\) stands for the output of the neural net. Let \(y\) be the labels of the training data. The learning process of the neural net is to minimize certain loss denoted by \(l(z, y) = l(f(x, w), y)\). We also denote \(l(f(x, w), y)\) by \(l(w)\) without unambiguity.

2.1 Natural Gradient

The gradient descent can be regarded as the direction in the tangent space of the parameter \(w\) that decreases the loss function at most. The well-know first-order gradient is the fastest direction with respect to the Euclidean \(l_2\) metric, and the basis of the most of gradient descent algorithms in practice, for instance, the SGD, Adam\[17\] and the others involved with momentums \[31\].

However, as argued by Amari \[2\], the \(l_2\) metric of the parameter’s tangent space in fact assumes that all parameters have the same weight in metric but does not take the characteristics of the neural net into considerations. In addition, this metric does not possess the parameter invariant property \[23\]. To conquer this issue, natural gradient of neural network was developed by Amari \[2\]. one of a general definition is

\[
\nabla_w^n l = \lim_{\epsilon \to 0} \frac{1}{\epsilon} \arg \min_{d, m(w, w+d) < \frac{\epsilon^2}{2}} (l(; w + d) - l(; w))
\]
where the metric is defined as $m(w, w + d) = l(f(x, w), f(x, w + d))$. Assuming (1), $l(z, z) = 0$, for all $z$; (2), $l(z, z') \geq 0$, for all $z$ and $z'$; (3), $l$ is differentiable with respect to $z$ and $z'$ which is true for the mean square loss and the cross-entropy loss, the metric $m(w, w')$ has the following expansion

$$m(w, w + d) = \frac{1}{2}d^\top H d + o(||d||^2_2), \quad H = \frac{\partial z}{\partial w}^\top H_1 \frac{\partial z}{\partial w}$$

(1)

where $\frac{\partial z}{\partial w}$ is the Jacobian matrix of $f(x, w)$ with respect to $w$ and $H_1 = \frac{\partial^2 z}{\partial z^2} l(z, z')|_{z=z'=f(x,w)}$ is the Hessian matrix of $l(z, z')$ with respect to $z$ when $z = z' = f(x, w)$. Hence, the natural gradient is specified as

$$\nabla_w^n l = \arg \min_{\|d\|_H = 1} \langle d, \frac{\partial l}{\partial w} \rangle = -\alpha H^{-1} \frac{\partial l}{\partial w}$$

(2)

where $\|d\|_H = \sqrt{d^\top H d}$ and $\alpha = 1/\|H^{-1} \frac{\partial l}{\partial w}\|_H$ is the normalization scalar. More specially, if $l(z, z')$ is the cross entropy loss, then $H$ is the Fisher information matrix, which is in agreement with the original definition in Amari [2].

In many applications, natural gradient performs much better than the gradient descent [24]. However, calculating the natural gradient in deep neural nets, has difficulty in practice, because calculating $H$ on a small mini-batch of the training data always causes $H$ of low ranks so that $H^{-1}$ does not exist in nature. One alternative is to use the damping technique [26, 20]: let $\bar{H} = H + \lambda I$, where $\lambda$ is a positive scalar. However, the selecting the proper value for the $\lambda$ is difficult: if $\lambda$ are too large, then natural gradient degenerates to the weighted gradient; if $\lambda$ is too small, the natural gradient could be too aggressive due to the low rank of $\bar{H}$ on a mini-batch of the training data.

### 2.2 Hessian Free Method in Neural Nets

Due to the arguments above, towards avoiding the calculating $H^{-1}$ directly, the Hessian free method was proposed by Martens [23]. Martens and Sutskever [26] to calculate nature gradient or other second-order gradient descent method in practice of deep neural nets. The key idea of Hessian free method is twofold: calculating $Hv$ and calculating $H^{-1}v$.

First, to calculate $Hv = \frac{\partial z}{\partial w}^\top H_1 \frac{\partial z}{\partial w}v$, we are to calculate (1), $\mu = \frac{\partial z}{\partial w}v$, (2), $u = H\mu$, and then (3).

$Hv = \frac{\partial z}{\partial w}^\top u$. In a multiple-layered neuronal net, $\mu$ is computed by an iterative forward way. At layer $k$, let $q_k(i_k, w_k)$ be the map of layer $k$ with $w_k$ and $i_k$ the parameters and input of layer $k$, and $R_k$ be the output of layer $k$. We have the following iterative formula:

$$\frac{\partial R_{k+1}}{\partial w}v = \frac{\partial R_k}{\partial w} \frac{\partial w_k}{\partial w}v + \frac{\partial R_k}{\partial i_k} \frac{\partial i_k}{\partial w}v,$$

(3)

noting $\frac{\partial w_k}{\partial w}v = w_k$, where $w_k$ be the partial of $v$ associated with $w_k$ in $w$, and $\frac{\partial i_k}{\partial w}v = \frac{\partial R_{k-1}}{\partial w}v$ in a BP layer, zero at the input layer, and also possibly has other formulas in other types of layer, for instance, the residual layer. Iteration of Equation (3) until the last output layer, i.e., $R_k = z$, gives $\mu$.

In addition, $u = H\mu$ is easy when $H_1$ is of low rank and $(Hv)^\top = u^\top \frac{\partial z}{\partial w}$ is a backward process. Also, this approach can be applied to $\bar{H}$, where $\bar{H}v = Hv + \lambda v$

Second, with an efficient calculation of $Hv$, the natural gradient $H^{-1}v$ can be approximated by the conjugate gradient method [13]. Algorithm [1] gives the pseudo-codes of the Preconditioned conjugate gradient (PCG) [4], where $P$ is the the Preconditioned Matrix, which is positive definite and usually takes a diagonal matrix, and $x_0$ is the initial value. It should be highlighted that the choice of $x_0$ and $P$ effects a lot on the convergence speed in practice.

The Hessian free method to train a neural net usually needs about 10 ~ 100 iterations of PCG for per training iteration of the neural net [24]. Therefore, this method possesses much more computation coast and so does not own any advantage in terms of the wall clock time in comparison to the first-order gradient method, for instance, the SGD, in particular, when training deep neural networks.

### 3 Meta-Learning with Hessian Free approach

To conquer the disadvantage of the Hessian free method but still remaining the advantage of the natural gradient, in this section, we propose a novel method of employing the meta-learning approach
Algorithm 1: Preconditioned conjugate gradient algorithm (PCG)

Aim : compute $A^{-1}b$

Inputs : $b, A$, initial value $x_0$, Preconditioned Matrix $P$, maximum iteration number $n$

$r_0 \leftarrow b - Ax_0$
$y_0 \leftarrow \text{solution of } P y = r_0$
$p_0 \leftarrow y_0; \ i \leftarrow 0$

while $\|r_i\|_2 \geq \text{threshold and } i \leq n$

$\alpha_i \leftarrow \frac{r_i^T y_i}{p_i^T Ap_i}$
$x_{i+1} \leftarrow x_i + \alpha_i p_i; \ r_{i+1} \leftarrow r_i - \alpha_i A p_i$
$y_{i+1} \leftarrow \text{solution of } P y = r_{i+1}$
$\beta_i+1 \leftarrow \frac{r_{i+1}^T y_{i+1}}{r_i^T y_i}$
$p_{i+1} \leftarrow y_{i+1} + \beta_{i+1} p_i$
$i \leftarrow i + 1$

end

Outputs : $x_n$ with $x_n \simeq A^{-1}b$, residual error $r_i$

Algorithm 2: Meta-Learning with Hessian Free Approach (MLHF)

Inputs : $n \leq 4$, learning rate $lr$, model $f$, loss function $l$

$d_n^0 \leftarrow 0; \ r_n^{-1} \leftarrow 0; \ t \leftarrow 0$
initialize parameters $w^0$

while not terminated do

get mini-batch input $x^t$ and label $y^t$
calculate $z^t = f(x^t, w^t)$ and $l^t = l(z^t, y^t)$
calculate gradient $g^t = \frac{\partial l^t}{\partial w^t}$
$d_n^0 \leftarrow d_n^{t-1}; \ r_n^t \leftarrow r_n^{t-1}$
$s^t \leftarrow \text{RNN}_s(d_n^0, r_n^0, g^t); \ P^t \leftarrow \text{diag}(\text{RNN}_p(d_n^0, r_n^0, g^t))$

def $H^t v = \frac{\partial z^t}{\partial w^t} H^t \frac{\partial z^t}{\partial w^t} v + s^t \odot v, \forall v$

$d_n^{t+1}, r_n^t \leftarrow \text{PCG}(g^t, H^t, d_n^0, P^t, n)$(see Algorithm 1)
$w^{t+1} \leftarrow w^t - lr \ast d_n^{t+1}$
t $\leftarrow t + 1$

end

Outputs : $w^t$
We utilize the Back-Propagation-Through-Time (BPTT) \cite{37} to meta-train RNN$_s$ and RNN$_p$ in parallel way, but with the different loss functions. Let $t = 1, \cdots, T$ be the iterative times of a sequence training process on target network in meta-training, the loss function of RNN$_p$ is

$$l_p = \frac{1}{T} \sum_t \frac{(d^t_n, g^t)}{\sqrt{(d^t_n, H^t d^t_n)}},$$

where $d^t_n$, $H^t$, $g^t$ are defined in Algorithm \cite{37}. It can be seen that minimizing $l_p$ can enhance the preciseness of estimation of the natural gradient by a few iterations of PCG. The loss function of RNN$_s$ is defined as

$$\begin{align*}
l_s^t &= l(f(x^{t+1}, w^{t+1}), y^{t+1}) + l(f(x^t, w^t), y^t) - 2 \times l(f(x^t, w^t), y^t), \quad (4) \\
l_s &= \sum_t l_s^t, \quad (5)
\end{align*}$$

Here $l_s^t$ is inspired by \cite{3} with some modifications by adding the second item $l(f(x^{t+1}, w^{t+1}), y^t)$ in formula \cite{4}. The motivation of this term comes from the challenge of meta-training that RNN has the tendency to predict the next input and to fit for it, but the mini-batch $x^t$ is indeed unpredictable in meta-training, which might cause overfitting or be hard to train at the early stage. Adding this item in \cite{4} can reduce such influence and so stabilize meta-training process. Thus, $l_s$ is the softmax weighted average over all $l_s^t$.

**Stop gradient propagation** In training RNN$_s$ and RNN$_p$, for the same consideration, we do not propagate the gradient of the meta-parameter through $w^t, g^t, d^t_0, r^t_0$ in Algorithm \cite{2} in the BPTT role-back, $l(f(x^t, w^t), y^t)$ of the third term in \cite{4}, and all $e^t_r$ in \cite{5}.

Another advantage of stopping back propagation of gradients of $w^t, g^t, d^t_0, r^t_0$ is to simplify the gradient of multiplication $Hv$ in PCG iterations. In detail, For $u = Hv$ (without the damping part), the $H$'s gradient in back-propagation progress is not conducted. For the gradient of $v$, we can get $\frac{\partial H}{\partial v} = H \frac{\partial v}{\partial v}$, that is the gradient operator of $H$ is it’s self. By this technique the calculation of the second-order gradient in meta-training is not necessarily any more, which also reduce GPU memory usage and simplify the calculation flow graph in practice.

**Experiment replay of $w$** During the meta-training, the inputs of one iterate in Algorithm \cite{2} contains $d^{-1}_m, r^{-1}_m$, $w^0$, and $\{(x^t, y^t)\}_{0 \leq t < T}$. $\{(x^t, y^t)\}_{0 \leq t < T}$ is sampled from the dataset; $d^{-1}_m$ and $r^{-1}_m$ take values of zeros in practice. But for the $w^0$, the common choice, random generating is not suitable, especially for the complex neural net such as ResNet \cite{14}. Here, we use the experiment replay technique \cite{28,32} to store and replay $w^0$, as shown in Algorithm \cite{3}.

\begin{algorithm}
\caption{Algorithm 3: experiment replay of $w^0$}
\begin{algorithmic}
\State **Inputs**: initial $l_{top}$
\State initialize parameters $w^0$
\State store($w^0$)
\While {not terminated}
\State $w^0 \leftarrow$ sample replay for repository; \quad get $\{(x^t, y^t)\}_{0 \leq t < T}$
\State $d^{-1}_m \leftarrow 0$; \quad $r^{-1}_m \leftarrow 0$
\State \textbf{meta-training steps} \ldots
\If {$t^{-1} < l_{top}$}
\State store($w^k$); \quad $l_{top} \leftarrow t^{-1}$
\EndIf
\EndWhile
\end{algorithmic}
\end{algorithm}

\footnote{Another nature choice is to minimize the square of the norm of $r_m$ in PCG, which means $l_p = \frac{1}{T} \sum_t ||r^t_m||_2^2$, but it seems not as good as using formula \cite{2}, considering that $||r^t_m||_2^2$ has a different scale and is hardly to be stable trained in the initial phase of meta-training.}
3.2 Analysis of Computation Complexity

The major time consumption of computation with the MLHF method is the forward (including the difference forward and the first common forward, while difference forward is much faster than the first common forwards considering that it can share intermediate result between forward in different times) and backward processes, other than the inference of the RNN. So the time complexity is $O(nK)$, where $n$ is the max iterations in PCG and $K$ is the time that finish one forward and backward process. Here, we set $n = 4$, which usually cause $1 - 2$ times as long as the SGD for each iteration, which is illustrated in Section 4.

4 Experiments

In experiments, we realize the MLHF method of Algorithm 2 by Tensorflow [11]. Here, RNNs and RNNp are set as two-layered LSTM [16] with tanh(·) as the preprocess, and a linear map following softplus as the post-process with each layer 4 units. In the meta-training process, the roll-back length of BPTT is set to 10. We use Adam as the optimizer for meta-training of RNNs, and the maximum number of iterations of PCG $n$ is fixed to 4 by default if without specification.

In the first and second experiments, we evaluate the MLHF on a simple model (CUDA-convnet) and a more complex one (ResNet18(v2)) in contrast with other optimizers, including gradient-based first-order optimizers, i.e. RMSprop, adam, SGD + momentum (noted as SGD(m)) and practical second-order optimizer kfac [25] [13]. In particular, for ResNet18(v2), we do not compare the MLHF with the kfac because realizing kfac on ResNet18 (v2) is out of the limitation of GPU memory. All optimizer’s hyper-parameter will be keep to default in Tensorflow without specification. The results of these two experiments are illustrated by the loss function (cross-entropy of the neural net $f(x, w)$) with respect to both the number of trained samples and wall time respectively. All the experiments were done on a single Nvidia GTX Titan Xp, and the code can be viewed in https://www.github.com/ozzzp/MLHF.

In this work, we do not include other meta-optimizers, i.e. L2L [3], to comparison, because we can show our method is superior even in a simple MLP and L2L is failed to be trained with efficient descent of the loss functions on CUDA-convnet by us. See the supplementary materials for details.

4.1 Experiment 1: Convnet on Cifar10

CUDA-Convnet [18] is a simple CNN with 2 convolution layers and 2 fully connect layers. Here, we use the variant of CUDA-Convnet, which drops off the LRN layer and uses the fully connect layer instead of local connected layer on the top of the model. We meta-train a MLHF optimizer with batch size equal to 64 by BPTT on cifar10 [19] for 250 epochs. After meta-training, we validate this meta-trained optimizer as well as the compared optimizers by training the same model on the same dataset with batch size of 128. Even though this model is quite simple, it has 186k parameters, which is indeed more than the previous models involved in learning-to-learn literature.

Figure 1(a) and (b) shows the MLHF optimizer performs much better than kfac and RMSprop, Adam, and SGD(m) in both sample number and wall time.

4.2 Experiment 2: Resnet on ILSVRC2012

To validate the generalization of MLHF between different datasets and different-but-similar neural network architectures, we realize a mini version of ResNet [14] model on cifar10 for 250 epochs, which has 9 res-block with channel [16, 16, 16, 32, 32, 32, 64, 64, 64], to meta-training. Then we employ the meta-trained MLHF to train a ResNet18(v2) on ILSVRC2012 [9] dataset. The batch size in meta-training is 128, and 64 in training ILSVRC2012, due to the limitation of GPU memory.

As shown in Figure 2 (a), the performance of MLHF to train the Resnet (v2) on ILSVRC2012 is the best of all evaluated optimizers, in terms of both of the rapid-descent early stage and the steady-descent later stage, counting by the training sample number. However, Figure 2 (b) indicates that the SGD(m) method performed as good as the MLHF at the early stage but and iterating faster than MLHF in wall time. It has also been seen that the MLHF has effective descent progress of the loss function during the whole long-time training, which overcomes the major shortcoming of the previous meta-learning methods [38].
Figure 1: Performance (cross-entropy) of the training processes of MLHF compared with other optimizers on the CUDA-Convnet model of dataset cifar10 for 250 epochs. The learning rate was set to 1e-3 for Adam, RMSprop and SGD + momentum which was the optimal choice in \{1e-1, 1e-2, 1e-3\}. For kfac, the learning rate was set to 1e-1 for the same optimal process, while damping parameter $\lambda$ was set to 1e-3 and mini-batch size was set to 512 as the Grosse and Martens [13] did. Other optimizer’s batch size is 128.

Figure 2: Performance of the training processes of MLHF with other optimizers on ResNet18 (v2) on the dataset ILSVRC2012. The learning rate was set to 1e-2 $\times \frac{b}{256}$ for Adam and RMSprop, and the 1e-1 $\times \frac{b}{256}$ for SGD(m) which was selected by random search, where $b$ is the batch-size and set to 64 for all optimizer.

4.3 Experiment 3: Ablation experiment

In this experiment, to verify the efficiency of RNN$_p$ towards the natural gradient, we employ the same meta-training configuration as in section 4.2 and conduct the following four configuration and re-meta-training the MLHF for contrast: (1). remove RNN$_p$ and set the maximum iteration number of PCG to 20; (2). remove RNN$_p$ and set the maximum iteration number of PCG to 4; (3). keep RNN$_p$ but set the maximum iteration of PCG to 2; (4). keep all to the default. We highlight that config (1) can be regarded as the best performance of PCG with a big cost of computation time.

From Figure 3(a) and (b), one can have the following observations. First, with the help of RNN$_p$, a very few (4) iterations of PCG (config 4) can estimate the natural gradient as precise as sufficient iterations of PCG (config 1) measured by $l_p$ (Figure 3(a)); however, 4 iterations is far away from convergence of PCG, in contrast, 20 iterations (config 1) can guarantee a good convergence of PCG, measured by the mean of $\|r_n\|^2$ (Figure 3(b)). Second, in contrast, without RNN$_p$, a few iterations of PCG (config 2) results in a bad estimation of natural gradient and of course far away from convergence of PCG. Finally, we highlight that 4 iterations could be the optimal number for PCG with the help of RNN$_p$, because further reduction of the number of iteration, i.e., 2 iterations of PCG (config 3), results in both a bad approximation of natural gradient and a bad convergence of PCG.
Figure 3: Ablation contrast results of $l_p$ (a) and $\frac{1}{T}\sum_t \|r_n\|^2_2$ (b) with respect to iterations for the underlying four configurations of the MLHF.

5 Conclusions and Discussions

In conclusion, we introduced a novel second-order meta-optimizer based on the Hessian Free approach. We utilized the PCG algorithm to approximate the natural gradient as the optimal descent direction for neural net training. By the coordinatewise framework, we designed RNN_s and RNN_p to infer the damping parameters and preconditioned matrix such that a very few number of iteration of PCG algorithm can achieve a good approximation of the natural gradient with an acceptably low computation cost. Furthermore, a few techniques were used to efficiently meta-train the MLHF. Then, experiments showed that this meta-optimizer can efficiently make progress during both the early and later stages of the whole long-time train process in a large-scale neuron nets with big datasets, including the CUDA-convnet on cifar10 and resnet18 (v2) on ILSVRC2012.

One explanation of this advantage of the MLHF is twofold. First, one can observe that however RNN_s trained, only if RNN_p works well in terms that $d_n$ approaches $(H + \text{diag}(s))^{-1}g$ well, we have $\langle d_n, g \rangle \simeq g^T (H + \text{diag}(s))^{-1}g > 0$, which implies that under any over-fitting scenario of RNN_s, the loss of $f(x, w)$ decreases with a sufficiently small learning rate. Therefore, the training process can be efficiently progressing even in the gradual stage. Second, it can be seen that each coordination of $d_n$ is determined by the whole $s$ and $P$, which may result in a good error-tolerance.

To sum up, this advantage implies that the presented meta-optimizer can be a promising meta-learning framework towards elevating the training efficiency in practical deep neural nets.

The limitation of this work still exists on the cost of wall time in comparison to the first-order gradient method. As the increases of the number of neural net parameters, the wall time cost of the meta-optimizer increases in proportion, that will weaken the superiority of training efficiency in a very large-scale neural net, given the computation resource.

For the future work, we wish to evaluate MLHF’s generalization on more extensive neural networks, including RNN, RCNN[11], etc, and develop the distributed version of MLHF (It is a little sad for us that did not accomplish the experience on ResNet50, cause that without distributed version, the maximum batch size on a single Nvidia GTX Titan Xp is only 8 that is too small to train ILSVRC2012). The simplification and accelerating is also one of orients. We have great expectation of this orient that can make learning-to-learn approach exhibit it’s promised efficacy in deep neural networks.

References

[1] M. i. n. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In OSDI, volume 16, pages 265–283, 2016.
[2] S.-I. Amari. Natural gradient works efficiently in learning. Neural computation, 10(2):251–276, 1998.
[3] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, and N. de Freitas. Learning to learn by gradient descent by gradient descent. In Advances in Neural Information Processing Systems, pages 3981–3989, 2016.
[4] K. E. Atkinson. *An introduction to numerical analysis*. John Wiley & Sons, 2008.

[5] S. Bengio, Y. Bengio, J. Cloutier, and J. Gecei. On the optimization of a synaptic learning rule. In *Preprints Conf. Optimality in Artificial and Biological Neural Networks*, pages 6–8. Univ. of Texas, 1992.

[6] S. Bengio, Y. Bengio, and J. Cloutier. On the search for new learning rules for anns. *Neural Processing Letters*, 2(4):26–30, 1995.

[7] Y. Bengio, S. Bengio, and J. Cloutier. *Learning a synaptic learning rule*. Université de Montréal, Département d’informatique et de recherche opérationnelle, 1990.

[8] Y. Chen, M. W. Hoffman, S. G. o. m. Colmenarejo, M. Denil, T. P. Lillicrap, M. Botvinick, and N. de Freitas. Learning to learn without gradient descent by gradient descent. *arXiv preprint arXiv:1611.03824*, 2016.

[9] J. Deng, A. Berg, S. Satheesh, H. Su, A. Khosla, and L. Fei-Fei. *Ilsvrc-2012*, 2012. URL http://www.image-net.org/challenges/LSVRC, 2012.

[10] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*, 2017.

[11] R. Girshick. Fast r-cnn. *arXiv preprint arXiv:1504.08083*, 2015.

[12] A. Graves, G. Wayne, and I. Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[13] R. Grosse and J. Martens. A kronecker-factored approximate fisher matrix for convolution layers. In *International Conference on Machine Learning*, pages 573–582, 2016.

[14] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[15] M. R. Hestenes and E. Stiefel. *Methods of conjugate gradients for solving linear systems*, volume 49. NBS Washington, DC, 1952.

[16] S. Hochreiter and J. u. r. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[17] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[18] A. Krizhevsky. cuda-convnet: High-performance c++/cuda implementation of convolutional neural networks. Source code available at https://github.com/akrizhevsky/cuda-convnet2 [March, 2017], 2012.

[19] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. 2009.

[20] Y. LeCun, L. e. o. Bottou, G. B. Orr, and K.-R. M ü ller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–50. Springer, 1998.

[21] K. Li and J. Malik. Learning to optimize. *arXiv preprint arXiv:1606.01885*, 2016.

[22] K. Li and J. Malik. Learning to optimize neural nets. *arXiv preprint arXiv:1703.00441*, 2017.

[23] J. Martens. Deep learning via hessian-free optimization. In *ICML*, volume 27, pages 735–742, 2010.

[24] J. Martens. New insights and perspectives on the natural gradient method. *arXiv preprint arXiv:1412.1193*, 2014.

[25] J. Martens and R. Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pages 2408–2417, 2015.

[26] J. Martens and I. Sutskever. Learning recurrent neural networks with hessian-free optimization. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1033–1040. Citeseer, 2011.

[27] J. Martens and I. Sutskever. Training deep and recurrent networks with hessian-free optimization. In *Neural networks: Tricks of the trade*, pages 479–535. Springer, 2012.

[28] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[29] D. K. Naik and R. Mammone. Meta-neural networks that learn by learning. In *Neural Networks, 1992. IJCNN., International Joint Conference on*, volume 1, pages 437–442. IEEE, 1992.

[30] S. Ravi and H. Larochelle. Optimization as a model for few-shot learning. 2016.

[31] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

[32] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[33] J. u. r. Schmidhuber. *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. PhD thesis, Technische Universität München, 1987.
[34] R. S. Sutton. Adapting bias by gradient descent: An incremental version of delta-bar-delta. In AAAI, pages 171–176, 1992.

[35] T. Tieleman and G. Hinton. Lecture 6.5-rmsprop, coursera: Neural networks for machine learning. University of Toronto, Technical Report, 2012.

[36] J. X. Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran, and M. Botvinick. Learning to reinforcement learn. arXiv preprint arXiv:1611.05763, 2016.

[37] P. J. Werbos. Backpropagation through time: what it does and how to do it. Proceedings of the IEEE, 78(10):1550–1560, 1990.

[38] O. Wichrowska, N. Maheswaranathan, M. W. Hoffman, S. G. Colmenarejo, M. Denil, N. de Freitas, and J. Sohl-Dickstein. Learned optimizers that scale and generalize. arXiv preprint arXiv:1703.04813, 2017.