Momentum acceleration of quasi-Newton based optimization technique for neural network training

Shahrzad Mahboubi1a), Indrapriyadarsini S2, Hiroshi Ninomiya1, and Hideki Asai3

1 Graduate School of Electrical and Information Engineering, Shonan Institute of Technology
1-1-25 Tsujido-Nishikaigan, Fujisawa, Kanagawa, 251-8511, Japan
2 Graduate School of Science and Technology, Shizuoka University
3-5-1 Johoku, Naka-ku, Hamamatsu, Shizuoka, 432-8011, Japan
3 Research Institute of Electronics, Shizuoka University
3-5-1 Johoku, Naka-ku, Hamamatsu, Shizuoka, 432-8011, Japan

a) 20T2502@sit.shonan-it.ac.jp

Received January 10, 2021; Revised March 12, 2021; Published July 1, 2021

Abstract: This paper describes a momentum acceleration technique for quasi-Newton (QN) based neural network training and verifies its performance and computational complexity. Recently, Nesterov’s accelerated quasi-Newton method (NAQ) has been introduced and shown that the momentum term is effective in reducing the number of iterations and the total training time by incorporating Nesterov’s accelerated gradient into QN. However, the gradients had to be calculated two times in one iteration in the NAQ training. This increased the computation time of a training loop compared with the conventional QN. The proposed technique is an improvement to NAQ done by approximating the Nesterov’s accelerated gradient as a linear combination of the current and previous gradients. As a result, the gradient is calculated only once per iteration similar to that of QN. The performance of the proposed algorithm is evaluated in comparison to conventional algorithms in neural networks training on two types of problems - function approximation problems with high nonlinearity and classification problems. The results show a significant acceleration in the computation time without losing the quality of the solution compared with conventional training algorithms.

Key Words: neural networks, training algorithm, quasi-Newton method, Nesterov’s accelerated gradient, momentum method

1. Introduction
Neural networks (NNs) have shown to be effective in numerous real-world applications [1, 2]. For example, NNs are efficiently used in solving the function approximation problems such as microwave
circuit modeling [3–5] which are problems with high nonlinear input-output characteristics or classification problems such as image classification [6] and the natural language processing [7]. Most of these applications require large NN models with massive amounts of training data to achieve good accuracies and low errors in minimum time. NN optimization poses several challenges such as ill-conditioning, vanishing, and exploding gradients, choice of hyperparameters, large computational cost, etc. Thus, the choice of the optimization algorithm is the most important step in developing NN’s models. In this research, the optimization of the following error function \( E(w) \) is considered for the training of the NN.

\[
\min_{w \in \mathbb{R}^n} E(w) = \frac{1}{|T_r|} \sum_{p \in T_r} E_p(w),
\]

where \( w \) is the weight vector and \( E_p(w) \) denotes the error function of the \( p^{th} \) training sample. Commonly used error functions are the mean squared error and the cross entropy error. \( T_r \) denotes a training data set and \(|T_r|\) is the number of training samples. Gradient based algorithms have been commonly used for training NNs. Most gradient based algorithms minimize (1) using the following iterative formula and the gradient of the error function \( \nabla E(w) \).

\[
w_{k+1} = w_k + v_{k+1},
\]

and

\[
v_{k+1} = -\eta_k \nabla E(w_k),
\]

where \( k \) is the iteration count and \( v_{k+1} \) is the update vector which is defined for each gradient algorithm with the parameter \( \eta_k \). Gradient based algorithms can be divided into two categories based on the form of \( \eta_k \): first-order methods and second (or super linear)-order methods [2]. First-order methods are often used because of their use of a scalar learning rate for \( \eta_k \) and low computational complexity of \( v_{k+1} \). A popular first-order algorithm is the gradient descent method (GD) or the so-called first error back-propagation algorithm [2]. \( \eta_k \) of GD is calculated by a line search strategy or set to a fixed value. Algorithms such as AdaGrad [8], AdaDelta [9] and RMSprop [10] have an adaptive \( \eta_k \) which is determined by an exponentially decaying average of past squared gradients. Other first-order algorithms include accelerated algorithms such as classical momentum method (CM) [11, 12], Nesterov’s accelerated gradient method (NAG) [11, 12] and Adam [13] which add an inertia or momentum term \( \mu_k v_k \) in (2) where \( \mu_k \) is the momentum coefficient. It is notable that, Adam is the most famous and effective first-order algorithm with adaptive \( \eta_k \) and momentum acceleration scheme. However, when applied to function approximation problems with high nonlinearity, these first-order methods still converge too slowly and optimization error cannot be effectively reduced within finite time in spite of its advantage [14].

Typically, second-order methods have been used to deal with this problem despite the increase in computational cost [1, 14–19] caused by the curvature information of the error function like Hessian or the approximated Hessian computation. In second-order methods, \( \eta_k \) of (3) is determined using a scalar \( \alpha_k \) and a matrix \( H_k \) as \( \alpha_k H_k \) unlike first-order methods that used a scaler. This paper uses the notation \( \eta_k \) as \( \eta_k H_k = \alpha_k H_k \) for second-order methods. Note that, \( H_k \) is defined as the approximated inverse Hessian in quasi-Newton method (QN). QN which is one of the most efficient second-order optimization algorithms has been widely used as a robust training algorithm [4, 15] for problems with highly nonlinear properties. Furthermore, several improvements for QN have been proposed in the past. One of the improvements is the Multi-step quasi-Newton method (MSQN) which uses the parameters and gradients in the past training steps for updating the curvature information [20, 21]. On the other hand, the acceleration of QN using the momentum term was introduced as Nesterov’s accelerated quasi-Newton method (NAQ) [14]. NAQ succeeded to drastically reduce the number of iterations and the total computational time compared to QN. However, NAQ has a disadvantage of calculating two gradients i.e. Nesterov’s accelerated gradient \( \nabla E(w_k + \mu_k v_k) \) and the normal gradient \( \nabla E(w_k) \) in one iteration while QN required only one gradient calculation per iteration.

In this research, a new quasi-Newton based training algorithm - Momentum quasi-Newton method (MoQ) - is introduced showing the performance for not only the function approximation problems
but also the classification ones. Actually, our MoQ was proposed and its effectiveness for only the function approximation problems was confirmed in [16]. This paper extends the validity of MoQ for neural network training using classification problems. Further, we also analyze the computational cost of MoQ compared to the conventional second-order methods. Note that, from the viewpoint of the error functions, the effectiveness of MoQ for both the mean squared and cross entropy errors which are commonly used in neural network training is confirmed by the performance of two types of problems.

The proposed MoQ accelerates NAQ by the approximation of Nesterov’s accelerated gradient vector as a linear combination of the current and previous normal gradients by assuming that the error function is approximately quadratic function around the current position. This approximation shows a reduction in the computation time required for one iteration compared with NAQ while maintaining the quality of the solution and fewer iterations. This paper verifies the performance of MoQ on several benchmark problems such as a function approximation problem with high nonlinearity, two real-world problems of microwave circuit modeling, and two classification problems. Through the results obtained by computer simulations, we show that the proposed method is suitable for a wide range of applications and problems for neural networks.

2. Related work

This section briefly shows the QN based training algorithm and its accelerated methods. These methods have been commonly used as the training algorithms for problems with highly nonlinear properties [4, 5, 14].

2.1 Quasi-Newton method (QN)

QN is one of the most efficient optimization algorithms using the curvature information [15, 17]. The learning rate $\eta_k$ in (3) is defined by the stepsize $\alpha_k$ (scalar) and the approximated inverse Hessian, $H_k^{QN}$ as $\eta_k H_k^{QN} = \alpha_k H_k^{QN}$. Then the update vector of QN is

$$v_{k+1} = -\alpha_k H_k^{QN} \nabla E(w_k),$$

where $H_k^{QN}$ is the symmetric positive definite matrix and is iteratively updated by the following BFGS (Broyden-Fletcher-Goldfarb-Shanno) formula [15],

$$H_{k+1}^{QN} = (I - (s_k y_k^T / s_k y_k^T)) H_k^{QN} (I - (y_k s_k^T / s_k y_k^T)) + (s_k s_k^T / s_k y_k^T),$$

where $s_k = w_{k+1} - w_k$ and $y_k = \nabla E(w_{k+1}) - \nabla E(w_k)$. Note that, the symmetric positive definiteness of $H_k^{QN}$ in the update of (5) is guaranteed [15, 17]. This means that the update vector of QN given by (4) and (5) always decreases $E(w)$. The stepsize $\alpha_k$ is determined according to Armijo’s condition [15] in (6).

$$E(w_k + \alpha_k g_k) \leq E(w_k) + \lambda \alpha_k \nabla E(w_k)^T g_k,$$

where $0 < \lambda < 1$ and was set to $\lambda = 10^{-3}$ in this paper. $g_k$ denotes the direction vector of each algorithm and $g_k = H_k^{QN} \nabla E(w_k)$ in QN. The algorithms of stepsize calculation and QN are shown in Algorithm 1 and Algorithm 2, respectively. Note that, the gradient is calculated only once per iteration as shown in step 4(c) in the training loop of Algorithm 2.

**Algorithm 1: Calculation of stepsize**

1. Initialize $\alpha_k = 1.0$, $i = 0$ and $ls = 10$
2. for $i < ls$ do
   (a) if Armijo’s condition is satisfied
   (b) break
   (c) else
   (d) Update $\alpha_k = \frac{1}{2} \alpha_k$
   (e) $i = i + 1$
3. return $\alpha_k$
NAQ is derived from the quadratic approximation of \( E(w) \) at \( w + \mu_k v_k \) as (7) using \( \Delta w = w - (w + \mu_k v_k) \) whereas QN approximated \( E(w) \) at \( w_k \).

\[
E(w) \simeq \hat{E}(w) = E(w_k + \mu_k v_k) + \nabla E(w_k + \mu_k v_k)^T \Delta w + (1/2)\Delta w^T \nabla^2 E(w_k + \mu_k v_k) \Delta w,
\]

where \( \nabla^2 E(w_k + \mu_k v_k) \) is the Hessian matrix of \( E(w) \) at \( w_k + \mu_k v_k \). The new iterate derived from \( \partial \hat{E}(w)/\partial w = 0 \), is defined as

\[
w_{k+1} = (w_k + \mu_k v_k) - \nabla^2 E(w_k + \mu_k v_k)^{-1} \nabla E(w_k + \mu_k v_k).
\]

This iteration is considered as the Newton method with momentum term \( \mu_k v_k \). Here the inverse of the Hessian is approximated by the matrix \( H_{k+1}^{NAQ} \) using

\[
H_{k+1}^{NAQ} = (I - (p_k^T q_k / p_k^T q_k))H_k^{NAQ} - \alpha_k (p_k^T / p_k^T q_k) + (p_k^T / p_k^T q_k),
\]

where \( p_k = w_{k+1} - (w_k + \mu_k v_k) \) and \( q_k = \nabla E(w_{k+1}) - \nabla E(w_k + \mu_k v_k) \). Equation (9) is derived from the Secant condition [15] \( q_k = (H_k^{NAQ} - 1)p_k \) and the rank-2 updating formula [14]. Note that, it was proved that the matrix \( H_{k+1}^{NAQ} \) updated by (9) holds the symmetric positive definite property if \( H_k^{NAQ} \) is a symmetric positive definite matrix. The update vector of NAQ is defined as

\[
v_{k+1} = \mu_k v_k - \alpha_k H_k^{NAQ} \nabla E(w_k + \mu_k v_k),
\]

where \( 0 < \mu_k < 1 \) and is usually chosen to be a value close to 1 such as 0.8, 0.85, 0.9 or 0.95 [11, 14]. NAQ drastically improved the convergence speed of QN using the momentum term \( \mu_k v_k \) and the gradient vector \( \nabla E(w_k + \mu_k v_k) \) called Nesterov’s accelerated gradient vector [11, 14]. This indicates that the momentum term is effective in acceleration of QN. The stepsize \( \alpha_k \) is determined according to Armijo’s condition for NAQ [14] as shown in (11).

\[
E(w_k + \mu_k v_k + \alpha_k g_k) \leq E(w_k + \mu_k v_k) + \lambda \alpha_k \nabla E(w_k + \mu_k v_k) - g_k,
\]

where \( g_k = H_k^{NAQ} \nabla E(w_k + \mu_k v_k) \). The algorithm of NAQ is shown in Algorithm 3. From this algorithm, it can be observed that there are two calculations of gradients \( \nabla E(w_k + \mu_k v_k) \) and \( \nabla E(w_{k+1}) \) shown in steps 3(b) and 3(f). This was a disadvantage of NAQ, but the algorithm could further shorten the iteration counts to cancel out the effect of the shortcoming compared with QN [14].

---

Algorithm 2: quasi-Newton method (QN)
1. \( k = 1 \)
2. Initialize \( w_k = \text{random}[-0.5, 0.5], H_k^{QN} = I \) (unit matrix)
3. Calculate \( \nabla E(w_k) \)
4. While(||\nabla E(w_k)|| > \epsilon \text{ and } k < k_{\text{max}})
   a. Calculate stepsize \( \alpha_k \) using Algorithm 1 and Armijo’s condition of (6)
   b. Calculate \( v_{k+1} = -\alpha_k H_k^{QN} \nabla E(w_k) \)
   c. Update \( w_{k+1} = w_k + v_{k+1} \)
   d. Calculate \( \nabla E(w_{k+1}) \)
   e. Update \( H_{k+1}^{QN} \) using (5)
   f. \( k = k + 1 \)
5. return \( w_k \)

2.2 Nesterov’s accelerated quasi-Newton method (NAQ)
NAQ is derived from the quadratic approximation of \( E(w) \) at \( w_k + \mu_k v_k \) as (7) using \( \Delta w = w - (w_k + \mu_k v_k) \) whereas QN approximated \( E(w) \) at \( w_k \).

\[
E(w) \simeq \hat{E}(w) = E(w_k + \mu_k v_k) + \nabla E(w_k + \mu_k v_k)^T \Delta w + (1/2)\Delta w^T \nabla^2 E(w_k + \mu_k v_k) \Delta w,
\]

where \( \nabla^2 E(w_k + \mu_k v_k) \) is the Hessian matrix of \( E(w) \) at \( w_k + \mu_k v_k \). The new iterate derived from \( \partial \hat{E}(w)/\partial w = 0 \), is defined as

\[
w_{k+1} = (w_k + \mu_k v_k) - \nabla^2 E(w_k + \mu_k v_k)^{-1} \nabla E(w_k + \mu_k v_k).
\]
Algorithm 3: Nesterov’s accelerated quasi-Newton method (NAQ)

1. \( k = 1 \)
2. Initialize \( w_k = \text{random}[-0.5, 0.5], H_k^{\text{NAQ}} = I \) (unit matrix) and \( v_k = 0 \)
3. While (\(||\nabla E(w_k)|| > \epsilon \) and \( k < k_{\text{max}} \))
   (a) Update \( \mu_k \)
   (b) Calculate \( \nabla E(w_k + \mu_k v_k) \)
   (c) Calculate stepsizes \( \alpha_k \) using Algorithm 1 and Armijo’s condition of (11)
   (d) Calculate \( v_{k+1} = \mu_k v_k - \alpha_k H_k^{\text{NAQ}} \nabla E(w_k + \mu_k v_k) \)
   (e) Update \( w_{k+1} = w_k + v_{k+1} \)
   (f) Calculate \( \nabla E(w_{k+1}) \)
   (g) Update \( H_{k+1}^{\text{NAQ}} \) using (9)
   (h) \( k = k + 1 \)
4. return \( w_k \)

2.3 Adaptive NAQ method (AdaNAQ)
NAQ has drastically improved the computing time but the momentum coefficient \( \mu_k \) which was fixed during the training had to be chosen carefully in order to obtain a stable and fast solution. The adaptive momentum coefficient \( \mu_k \) introduced in [12] was applied to NAQ as Adaptive NAQ (AdaNAQ) in [18, 19]. In this method, the adaptive formula of \( \mu_k \) is given by

\[
\mu_k = \theta_k (1 - \theta_k) / \theta_k^2 + \theta_k + 1, \tag{12}
\]

\[
\theta_{k+1}^2 = (1 - \theta_{k+1}) \theta_k^2 + \gamma \theta_{k+1}. \tag{13}
\]

where \( \theta_0 = 1 \) and \( \gamma = 10^{-5} \). The \( \mu_k \) given by (12) gradually changes from 0 to 1 during the training with maintaining the stabilities of the training and fast convergence speed of NAQ.

3. Proposed algorithm - momentum quasi-Newton method (MoQ)
While QN calculates only the normal gradient per iteration of the training loop, NAQ involves two gradient calculations – Nesterov’s accelerated gradient \( \nabla E(w_k + \mu_k v_k) \) and the normal gradient \( \nabla E(w_{k+1}) \). In order to reduce the number of gradient computations per iteration without losing the performance of NAQ, we proposed a novel quasi-Newton based training algorithm called Momentum quasi-Newton method (MoQ) [16]. In this paper, we further test the limits of our proposed method by evaluating its performance on a variety of problems and analyze its computational cost. In the following section, the algorithmic details and convergence of MoQ are described.

3.1 Momentum quasi-Newton method (MoQ)
MoQ is realized by approximating Nesterov’s accelerated gradient vector as a linear combination of the current \((k)\) and previous \((k-1)\) normal gradients, thus making it possible to calculate only one gradient per iteration. MoQ approximates the error function \( E(w) \) by assuming that the function is approximately quadratic in the neighborhood of \( w_k + \mu_k v_k \). This assumption is similar to the second-order Taylor approximation of \( E(w) \) in (7) which is commonly used in the derivation of second-order optimization methods [15]. Note that \( E(w) \) is an actual quadratic function but \( E(w) \) is not. This implies that \( \nabla E(w_k + \mu_k v_k) \) is approximately linear as

\[
\nabla E(w_k + \mu_k v_k) \approx \nabla E(w_k) + \mu_k \nabla E(v_k). \tag{14}
\]

Furthermore, since \( v_k = w_k - w_{k-1} \), (14) can be rewritten as

\[
\nabla E(w_k) + \mu_k \nabla E(v_k) = \nabla E(w_k) + \mu_k \nabla E(w_k - w_{k-1})
= (1 + \mu_k) \nabla E(w_k) - \mu_k \nabla E(w_{k-1}). \tag{15}
\]

From (14) and (15), it is confirmed that Nesterov’s accelerated gradient can be approximated as an extrapolation of \( \nabla E(w_k) \) and \( \nabla E(w_{k-1}) \) with a momentum coefficient \( \mu_k \), that is, a weighted
linear combination [16]. In this paper, $\mu_k \nabla E(v_k) = \mu_k \{ \nabla E(w_k) - \nabla E(w_{k-1}) \}$ is referred to as momentum gradient (MoG) term. Therefore, MoQ can be regarded as a method accelerating QN using two momentum terms of $\mu_k v_k$ and $\mu_k \nabla E(v_k)$. The update vector $v_{k+1}$ of MoQ can be defined by approximating the $\nabla E(w_k + \mu v_k)$ in (10) as

$$v_{k+1} = \mu_k v_k - \alpha_k H^\text{MoQ}_k \{(1 + \mu_k)\nabla E(w_k) - \mu_k \nabla E(w_{k-1})\}. \tag{16}$$

The stepsizes $\alpha_k$ is determined by the Armijo’s condition as shown in (17).

$$E(w_k + \mu_k v_k + \alpha_k g_k) \leq E(w_k + \mu_k v_k) + \lambda \alpha_k \{ (1 + \mu_k)\nabla E(w_k) - \mu_k \nabla E(w_{k-1}) \}^T g_k,$$  

where $g_k = H^\text{MoQ}_k \{(1 + \mu_k)\nabla E(w_k) - \mu_k \nabla E(w_{k-1})\}$. The matrix $H^\text{MoQ}_k$ is updated by,

$$H^\text{MoQ}_{k+1} = (I - (p_k q_k^T/p_k q_k) H^\text{MoQ}_k (I - (q_k p_k^T/p_k q_k)) + (p_k p_k^T/p_k q_k^T q_k). \tag{18}$$

where the Secant condition $\hat{q}_k = (H^\text{MoQ}_k)^{-1} p_k$ for MoQ is satisfied and

$$p_k = w_{k+1} - (w_k + \mu_k v_k) = w_{k+1} - (1 + \mu_k) w_k + \mu_k w_{k-1}, \tag{19}$$

$$\hat{q}_k = \nabla E(w_{k+1}) - (1 + \mu_k) \nabla E(w_k) + \mu_k \nabla E(w_{k-1}). \tag{20}$$

The algorithm of MoQ is shown in Algorithm 4. From this algorithm, it can be observed that MoQ computes the gradient only once per iteration as in step 4(e). Note that, the multi-step QN (MSQN) [20, 21] was also an acceleration technique of QN that uses (19) and (20) for the update of MoQ computes the gradient only once per iteration as in step 4(e). Note that, the multi-step QN (MSQN) [20, 21] was also an acceleration technique of QN that uses (19) and (20) for the update of MoQ. However, MSQN only extends Secant condition using the past steps $w_i$ and $\nabla E(w_i)$, $(i = k-1, k-2, \ldots)$ and the update vector $v_{k+1}$ of MSQN is that same as that of QN in (4) but the approximated Hessian is updated by (18). Wheres, the proposed MoQ algorithm accelerates QN using the MoG term as shown in the update vector in (16) and the approximated Hessian is updated by (18).

**Algorithm 4: Momentum quasi-Newton method (MoQ)**

1. $k = 1$
2. Initialize $w_k = \text{random}[-0.5, 0.5]$, $H^\text{MoQ}_k = I$ (unit matrix) and $v_k = 0$
3. Calculate $\nabla E(w_k)$
4. While$(||\nabla E(w_k)|| > \epsilon$ and $k < k_{\text{max}}$)
   (a) Update $\mu_k$
   (b) Calculate stepsizes $\alpha_k$ using Algorithm 1 and Armijo’s condition of (17)
   (c) Calculate $v_{k+1} = \mu_k v_k - \alpha_k H^\text{MoQ}_k \{(1 + \mu_k)\nabla E(w_k) - \mu_k \nabla E(w_{k-1})\}$
   (d) Update $w_{k+1} = w_k + v_{k+1}$
   (e) Calculate $\nabla E(w_{k+1})$
   (f) Update $H^\text{MoQ}_{k+1}$ using (18)
   (g) $k = k + 1$
5. return $w_k$

3.2 Convergence of MoQ

It is shown that $H^\text{MoQ}_{k+1}$ which is derived from (18) holds the symmetric positive definite property under the assumption that $H^\text{MoQ}_k$ is a symmetric positive definite matrix. To be more specific, the following statements hold true:

(a) $H^\text{MoQ}_{k+1}$ of (18) satisfies the Secant condition of MoQ, $\hat{q}_k = (H^\text{MoQ}_{k+1})^{-1} p_k$.

(b) If $H^\text{MoQ}_k$ is symmetric, $H^\text{MoQ}_{k+1}$ is also symmetric.

(c) If $H^\text{MoQ}_k$ is a positive definite matrix, $H^\text{MoQ}_{k+1}$ is also a positive definite matrix.

**Proof of (a):**

From (18) and the Secant condition $\hat{q}_k = (H^\text{MoQ}_{k+1})^{-1} p_k$:

$$\left(H^\text{MoQ}_{k+1}\right)^{-1} p_k = \left(I - \frac{p_k q_k^T}{p_k q_k} H^\text{MoQ}_k \left(I - \frac{q_k p_k^T}{p_k q_k}\right) + \frac{p_k p_k^T}{p_k q_k^T q_k}\right)^{-1} p_k, \tag{21}$$
by applying the Sherman-Morrison-Woodbury formula [15] to (21),

$$\text{where } B_k^{\text{MoQ}} = \left( H_k^{\text{MoQ}} \right)^{-1},$$

$$(B_k^{\text{MoQ}} + (q_k q_k^T/\hat{q}_k^T p_k) - (B_k^{\text{MoQ}} p_k p_k^T B_k^{\text{MoQ}}/p_k^T B_k^{\text{MoQ}} p_k) ) p_k,$$

(22)

Proof of (b): This is clear from (18).

Proof of (c):
First, $q_k^T p_k > 0$ will be shown. When the stepsize $\alpha_k$ is calculated by the exact line search as

$$dE(w_{k+1})/\alpha_k = -\nabla E(w_{k+1})^T H_k^{\text{MoQ}} \{ \nabla E(w_k) + \mu_k \nabla E(v_k) \} = 0,$$

(23)

then,

$$\hat{q}_k^T p_k = \{ \nabla E(w_{k+1}) - (\nabla E(w_k) + \mu_k \nabla E(v_k)) \}^T \{ w_{k+1} - (w_k + \mu_k v_k) \}
= \{ \nabla E(w_{k+1}) - (\nabla E(w_k) + \mu_k \nabla E(v_k)) \}^T \{ -\alpha_k H_k^{\text{MoQ}} (\nabla E(w_k) + \mu_k \nabla E(v_k)) \}
= \alpha_k (\nabla E(w_k) + \mu_k \nabla E(v_k))^T H_k^{\text{MoQ}} (\nabla E(w_k) + \mu_k \nabla E(v_k)) > 0,$$

(24)

is derived where $\nabla E(w_k) + \mu_k \nabla E(v_k) = (1+\mu_k) \nabla E(w_k) - \mu_k \nabla E(w_{k-1})$ and $w_k + \mu_k v_k = (1+\mu_k) w_k - \mu_k w_{k-1}$. It is guaranteed in (24) that $H_k^{\text{MoQ}}$ is a positive definite matrix and $\nabla E(w_k) + \mu_k \nabla E(v_k) \neq 0$.

Second, let $r \neq 0$ be an arbitrary vector, $r^T B_k^{\text{MoQ}} r > 0$ will be shown. For simplicity of the proof, the positive definiteness of $B_k^{\text{MoQ}} = \left( H_k^{\text{MoQ}} \right)^{-1}$ will be shown. It is clear that if the positive definiteness of $B_k^{\text{MoQ}}$ is proved, its inverse $H_k^{\text{MoQ}}$ is also a positive definite matrix. Because $H_k^{\text{MoQ}}$ is a positive definite matrix, $B_k^{\text{MoQ}}$ is also a positive definite matrix and it can be decomposed as $B_k^{\text{MoQ}} = C C^T$ using some non-singular matrix $C$. Let $t = C^T r (\neq 0)$ and $u = C^T p_k (\neq 0)$, it is shown using (22) that

$$r^T B_k^{\text{MoQ}} r = \left( (t^T u) - (t^T u)^2 \right)/u^T u + (r^T q_k)^2/q_k^T p_k \geq 0,$$

(25)

with the Cauchy-Schwarz inequality [15] and the condition of (24). In (25) the equality condition is satisfied, if and only if $(t^T u) - (t^T u)^2 = 0$ and $r^T q_k = 0$. The former equation holds when $t = \gamma u$ with the arbitrary scalar $\gamma (\neq 0)$. When $t = \gamma u$, then $r = \gamma p_k$. Therefore the latter equation is transformed as $r^T q_k = \gamma p_k^T q_k = 0$. This contradicts (24). Then the equality condition of (25) is not satisfied. As a result, $B_k^{\text{MoQ}}$ holds $r^T B_k^{\text{MoQ}} r > 0$, implying positive definiteness. As a result, $H_k^{\text{MoQ}}$ is also symmetric positive definite.

Furthermore, in order to guarantee the numerical stability and the global convergence of MoQ, a global convergence term is incorporated in $\hat{q}_k$, as shown in (26), (27) and (28) [22]. In this paper, this term is applied to QN [22], MSQN [16], NAQ [23] and MoQ for fair comparison.

$$\hat{q}_k = \nabla E(w_{k+1}) - (1+\mu_k) \nabla E(w_k) + \mu_k \nabla E(w_{k-1}) + \xi_k p_k = \delta_k + \xi_k p_k,$$

(26)

where

$$\xi_k = \omega \| (1+\mu_k) \nabla E(w_k) - \mu_k \nabla E(w_{k-1}) \| + max \{ -\delta_k^T p_k/\| p_k \|^2, 0 \},$$

(27)

and

$$\omega = 2 \quad if \quad \| (1+\mu_k) \nabla E(w_k) - \mu_k \nabla E(w_{k-1}) \|^2 > 10^{-2},$$

$$\omega = 100 \quad if \quad \| (1+\mu_k) \nabla E(w_k) - \mu_k \nabla E(w_{k-1}) \|^2 < 10^{-2}.$$

(28)
3.3 Computation cost of MoQ
The summary of the computational cost is illustrated in Table I. The cost of the gradient evaluations can be considered to be $nd$, where $n$ is the number of training samples involved and $d$ is the dimension of parameters. The update cost of the approximated Hessian matrix is $d^2$. In QN, NAQ and MoQ algorithms, the step length is determined by line search methods which involve $\zeta$ function evaluations until the search condition is satisfied. NAQ computes the gradient twice per iteration while QN and MoQ compute the gradient only once per iteration. Therefore, NAQ has an additional $nd$ computation cost. The approximation implemented in MoQ reduces the number of gradient calculations per iteration to one. As a result, it can be considered that QN and MoQ have the same computation cost per iteration.

| Algorithm | Computational Cost |
|-----------|--------------------|
| QN        | $nd + d^2 + \zeta nd$ |
| NAQ       | $2nd + d^2 + \zeta nd$ |
| MoQ       | $nd + d^2 + \zeta nd$ |

4. Simulation results
This section discusses the performance of MoQ on five benchmark problems, i.e., three function approximation and two classification problems. For the function approximation problems, we use a benchmark problem with high nonlinearity [14] and two real-world problems such as the microwave circuit modeling [5, 24]. For classification problems, $8 \times 8$ MNIST handwritten digits [25–27] and three-spirals [28] datasets are used. Ten independent runs with $w$ initialized by uniform random numbers in the range $[-0.5, 0.5]$ are conducted. The stepsize $\alpha_k$ for GD and MSQN are determine by the Armijo’s conditions of (6) in which $\alpha_k = \eta_k$ and $g_k = \nabla E(w_k)$ for GD while $g_k = H_{k}^{\text{MSQN}} \nabla E(w_k)$ for MSQN. The simulations were performed on a AMD Ryzen 7 1700X system with 3.4GHz processor and 16GB RAM.

4.1 Function approximation problems
First, three highly nonlinear function modelings are conducted in order to demonstrate the validity of the proposed MoQ. In these problems, the error function $E(w)$ is given by the mean squared error (MSE) as shown in (29).

$$\min_{w \in \mathbb{R}^n} E(w) = \frac{1}{|T_r|} \sum_{p \in T_r} E_p(w), \quad (1)$$

where,

$$E_p(w) = \frac{1}{2} \|d_p - o_p\|^2, \quad (29)$$

where $o_p$ and $d_p$ are the $p^{th}$ output and desired vectors, respectively. In the simulations, fully connected feedforward neural networks with one or two hidden layers are considered and the number of hidden neurons is decided for each problem. Each hidden neuron has a sigmoid activation function as $\text{sig}(z) = 1/(1 + \exp(-z))$. The performance of MoQ is compared with conventional algorithms such as GD, AdaGrad, RMSprop, AdaDelta, Adam, QN, MSQN, NAQ and AdaNAQ. Note that, the first-order adaptive methods such as AdaGrad, AdaDelta, RMSprop and Adam are mainly used in the stochastic (mini-batch) mode in which a portion of $T_r$ is used for the calculation of the gradient in a training loop. However, the full-batch strategy is considered in this paper because the mini-batch method cannot get sufficiently small errors, especially in the problems with high nonlinearity [29]. The mini-batch scheme of the proposed MoQ is out of the scope of this paper. Therefore, the full-batch scheme is applied to all algorithms for all problems. The momentum coefficients $\mu_k$ used in NAQ and MoQ are set to fixed values $\{0.8, 0.85, 0.9$ and $0.95\}$ and the adaptive value as shown in (12). Each trained neural network is evaluated by the median, average, best and worst of $E(w)$, the average of computational time ($\text{sec}$) and the average of iteration count ($k$) for 10 trials. The training error is
denoted by $E_{\text{train}}(w)$. The testing error shown as $E_{\text{test}}(w)$ is also calculated by (29) using a test data set $T_e$ which is independent of the training data set $T_r$. Each element of the input and desired vectors of $T_r$ and $T_e$ are normalized in the range $[-1.0, 1.0]$ in the simulations. The termination conditions are set to $\epsilon = 1.0 \times 10^{-8}$ and $k_{\text{max}} = 150,000$. In the simulations, we consider the convergence rate ($\%$) for each algorithm. This shows the rate at which the condition of $||\nabla E(w_k)|| < \epsilon$ can be obtained within $k_{\text{max}}$.

4.1.1 Function modeling problem

First, (30) is used as a function modeling problem with high nonlinearity [14, 19, 23],

$$f(a, x) = 1 + (x + 2x^2)\sin(-ax^2).$$

This function has two inputs $x$ and $a$. The ranges for this simulation are $x \in [-4, 4]$ and $a \in [-1, 1]$. $T_r$ and $T_e$ include 3,320 training and 6,600 testing points, respectively. The trained network has a hidden layer with 55 neurons. Therefore, the structure of the neural network is 2-55-1. In this problem, the learning rates of AdaGrad, AdaDelta, RMSprop, and Adam were set to 0.01, 1.0, 0.1 and 0.1, respectively. Each value is decided by the best performance for each algorithm in our experiments. The results are summarized in Table II. From this table, it is can be observed that first-order methods such as GD, AdaGrad, AdaDelta, RMSprop and Adam could not find sufficiently small training and testing errors within $k_{\text{max}}$ whereas all QN based methods could obtain sufficient small errors. Furthermore, it was confirmed that the momentum terms were effective for faster convergence of QN based algorithms because the momentum based QN such as NAQ and MoQ could obtain small iteration counts and fast computation times compared with QN and MSQN. In the viewpoint of the comparison of NAQ with the proposed MoQ, the computation time of MoQ was much faster than NAQ although the iteration counts of MoQ was similar to those of NAQ. This confirms that (14) is a good approximation of the gradient at $w_k + \mu_k v_k$, resulting in a reduced number of gradient computations and faster training. However, for larger values of $\mu_k$ (0.9 and 0.95) it was observed that the average convergence rate of MoQ was less than 100\% due to overshooting of gradients in some trials. This problem was resolved in AdaMoQ using the adaptive scheme for $\mu_k$ as in (12) without losing the fast convergence of MoQ.

| Algorithm | $\mu_k$ | $E_{\text{train}}(w) \times 10^{-3}$ | Time (sec) | Iteration counts (k) | $E_{\text{test}}(w) \times 10^{-3}$ | Convergence Rate(%) |
|-----------|---------|-------------------------------------|------------|----------------------|---------------------------------|---------------------|
| GD        | -       | 42.31 / 42.29 / 42.24 / 42.31       | 578        | 150,000              | 41.34 / 41.33 / 41.26 / 41.35   | -                   |
| AdaGrad   | -       | 39.72 / 39.42 / 35.05 / 42.21       | 251        | 150,000              | 38.74 / 38.44 / 34.06 / 41.24   | -                   |
| AdaDelta  | -       | 76.50 / 71.23 / 42.38 / 92.93       | 250        | 150,000              | 70.27 / 70.27 / 41.42 / 91.98   | -                   |
| RMSprop   | -       | 33.24 / 33.12 / 32.24 / 33.34       | 250        | 150,000              | 32.40 / 32.28 / 31.41 / 32.49   | -                   |
| Adam      | -       | 3.92 / 6.83 / 1.80 / 25.17          | 253        | 150,000              | 3.67 / 6.57 / 1.72 / 24.49      | -                   |
| QN        | -       | 0.453 / 0.483 / 0.319 / 0.830       | 257        | 94,212               | 0.491 / 0.454 / 0.303 / 0.765   | 100                 |
| MSQN      | -       | 0.620 / 0.662 / 0.408 / 1.21        | 214        | 78,489               | 0.593 / 0.618 / 0.385 / 1.10    | 100                 |
| NAQ       | 0.8     | 0.567 / 0.861 / 0.336 / 3.85        | 192        | 44,348               | 0.534 / 0.822 / 0.317 / 3.70    | 100                 |
|           | 0.85    | 0.432 / 0.547 / 0.362 / 1.17        | 165        | 38,171               | 0.407 / 0.517 / 0.341 / 1.10    | 100                 |
|           | 0.9     | 0.393 / 0.382 / 0.223 / 0.500       | 132        | 30,427               | 0.372 / 0.361 / 0.213 / 0.473   | 100                 |
|           | 0.95    | 0.521 / 4.26 / 0.291 / 87.85        | 102        | 23,546               | 0.494 / 4.14 / 0.274 / 36.95    | 100                 |
| AdaNAQ    | -       | 0.442 / 0.459 / 0.268 / 0.679       | 120        | 27,701               | 0.421 / 0.434 / 0.254 / 0.611   | 100                 |
| MoQ       | 0.8     | 0.434 / 0.469 / 0.308 / 0.726       | 118        | 43,168               | 0.414 / 0.447 / 0.293 / 0.722   | 100                 |
|           | 0.85    | 0.516 / 3.01 / 0.327 / 26.06        | 119        | 43,640               | 0.407 / 2.94 / 0.308 / 25.55    | 100                 |
|           | 0.9     | 0.421 / 0.525 / 0.349 / 0.995       | 75         | 27,516               | 0.399 / 0.489 / 0.327 / 0.902   | 100                 |
|           | 0.95    | 0.471 / 0.730 / 0.328 / 2.66        | 80         | 29,204               | 0.443 / 0.639 / 0.313 / 2.55    | 80                  |
| AdaMoQ    | -       | 0.460 / 0.457 / 0.220 / 0.842       | 74         | 27,046               | 0.435 / 0.428 / 0.206 / 0.773   | 100                 |
with large fixed $\mu_k$. It is shown that the adaptive scheme of $\mu_k$ in AdaMoQ is also effective in reducing the computation time of AdaNAQ. For measuring the accuracy of modeling, the output of the neural model trained by AdaMoQ with the error of $E_{\text{train}}(w) = 0.220 \times 10^{-3}$ is compared with the function of (30) in Fig. 1. The input $a$ is fixed to $-1$. From the figure, it is confirmed that the function of (30) and the model show a good match between them.

### 4.1.2 Microwave circuit modeling of Low-Pass Filter

Next, MoQ is applied to develop a neural network model of a microstrip Low-Pass Filter (LPF) illustrated in Fig. 2 [5, 30]. The dielectric constant and height of the substrate of LPF are 9.3 and 1mm, respectively. The length $D$ ranges between 12-20mm at intervals of 1mm in which training data $T_r$ and $T_e$ are $D = [12, 14, 16, 18, 20]$mm and $[13, 15, 17, 19]$mm, respectively. Training data is illustrated in Fig. 3. The frequency range is 0.1 to 4.5GHz. Each set contains 221 samples. The numbers of samples $T_r$ and $T_e$ are 1,105 and 884, respectively. The inputs of the NN are length $D$ and frequency $f$. The outputs are $|S_{11}|$ and $|S_{12}|$ which are the magnitudes of S-parameters, $S_{11}$ and $S_{12}$. The NN has a hidden layer with 45 hidden neurons. The structure of the neural network is 2-45-2.

The training and testing data were generated using Sonnet, a full-wave 3D planar electromagnetic field solver software for high-frequency EM simulation [31]. The summary of results is shown in Table III. The first-order methods were excluded in this table because these algorithms could not obtain sufficiently small errors. Moreover, to investigate the effect of the momentum coefficient for the convergence rates in detail, the range of $\mu_k$ was expanded. From the table, we can observe the effect of momentum acceleration on QN based algorithms for this problem as well. However, NAQ and MoQ with fixed $\mu_k$ are much faster when $\mu_k$ becomes large but are unstable as they fail to converge for some trials because of the high nonlinearity of the problem compared to the function modeling problem in 4.1.1. These results were more pronounced in MoQ. On the other hand, AdaNAQ and AdaMoQ which incorporate the adaptive scheme of $\mu_k$ could overcome this instability and achieve a 100% average convergence rate. As a result, it is confirmed that the proposed AdaMoQ is robust without sacrificing the computational speed. The results are also evident in Figs. 4 and 5. Figure 4 shows the graph of

![Fig. 2. Layout of Microstrip Low-Pass filter (LPF).](image)
Table III. Summary of simulation results of LPF modeling.

| Algorithm | µ  | \(E_{\text{train}}(w) \times 10^{-3}\) | Time (sec) | Iteration counts (k) | \(E_{\text{test}}(w) \times 10^{-3}\) | Convergence Rate(%) |
|-----------|----|-------------------------------------|------------|----------------------|---------------------------------|------------------|
| QN        | -  | 0.772 / 0.763 / 0.627 / 0.911 | 120        | 106,287              | 0.640 / 0.901 / 0.534 / 2.01 | 100              |
| MSQN      | -  | 0.708 / 0.746 / 0.624 / 0.969 | 109        | 96,938               | 0.555 / 0.900 / 0.430 / 2.65 | 100              |
| NAQ       | 0.7 | 0.720 / 0.727 / 0.607 / 0.904 | 126        | 66,133               | 0.819 / 1.87 / 0.466 / 8.94  | 100              |
|           | 0.75 | 0.677 / 0.698 / 0.608 / 0.856 | 112        | 59,100               | 0.759 / 0.743 / 0.406 / 1.07 | 100              |
|           | 0.8 | 0.792 / 0.791 / 0.570 / 1.08  | 89         | 46,861               | 0.962 / 8.02 / 0.435 / 65.32 | 100              |
|           | 0.85 | 0.729 / 0.742 / 0.526 / 0.994 | 95         | 49,823               | 0.922 / 1.24 / 0.406 / 3.18  | 100              |
|           | 0.9 | 0.684 / 0.670 / 0.499 / 0.908 | 84         | 44,149               | 0.683 / 1.10 / 0.442 / 3.60  | 100              |
|           | 0.95 | 0.661 / 4.09 / 0.571 / 22.30  | 51         | 22,338               | 1.20 / 3.79 / 0.525 / 19.60  | 80               |
| AdaNAQ    | -  | 0.596 / 0.615 / 0.486 / 0.739 | 64         | 33,715               | 0.652 / 0.667 / 0.464 / 1.12 | 100              |
| MoQ       | 0.7 | 0.659 / 0.660 / 0.599 / 0.710 | 80         | 72,090               | 0.749 / 0.965 / 0.566 / 2.18  | 80               |
|           | 0.75 | 0.753 / 0.740 / 0.626 / 0.826 | 43         | 38,350               | 0.797 / 0.819 / 0.539 / 1.31  | 70               |
|           | 0.8 | 0.677 / 0.649 / 0.569 / 0.727 | 38         | 34,361               | 0.703 / 0.819 / 0.482 / 1.60  | 50               |
|           | 0.85 | 0.677 / 1.63 / 0.635 / 5.17  | 23         | 20,742               | 0.988 / 3.5 / 0.586 / 33.30  | 50               |
|           | 0.9 | 0.660 / 0.667 / 0.641 / 0.705 | 16         | 14,262               | 0.647 / 0.636 / 0.488 / 0.761 | 40               |
|           | 0.95 | 0.567 / 0.604 / 0.545 / 0.735 | 14         | 12,847               | 0.702 / 0.713 / 0.578 / 0.870 | 40               |
| AdaMoQ    | -  | 0.675 / 0.615 / 0.472 / 0.751 | 31         | 27,627               | 0.798 / 1.14 / 0.591 / 2.15  | 100              |

Fig. 3. Training data set for LPF.

the minimum training error of 10 trials vs. the number of iterations in the training of NNs with QN, MSQN, AdaNAQ and AdaMoQ. From Fig. 4, it can be observed that AdaMoQ can maintain the same performance as AdaNAQ. On the other hand, Fig. 5 shows the training error vs. time, where it can be seen that the proposed AdaMoQ is the fastest algorithm. For measuring the accuracy of modeling, the outputs of the neural model trained by AdaMoQ with \(E_{\text{train}}(w) = 0.472 \times 10^{-3}\) is compared with test data of \(D = 17\text{mm}\) in Fig. 6, and it shows a good match between them.

4.1.3 Microwave circuit modeling of microstrip patch antenna

The second microwave modeling problem is the rectangular microstrip patch antenna (MPA) [18, 19, 24], illustrated in Fig. 7. The inputs of the NN model of MPA are length \(L\), width \(W\), frequency \(f\) and \(R = \frac{x}{L}\) which is the percentage ratio of relative distance \(x\) of the microstrip feed to \(L\). The two outputs correspond to the real and imaginary parts of the S-parameter \(S_{11}\). The length \(L\) and the
Fig. 4. The training error vs. iteration counts of LPF.

Fig. 5. The training error vs. time of LPF.

Fig. 6. Comparison of test data vs. neural model trained by AdaMoQ.
width $W$ ranges were 10-30mm at intervals of 2mm. The range of frequency $f$ was 0.5 to 5.0GHz at intervals of 0.5GHz, and the range of $R$ was 10% and 15% to 85% at intervals of 10%. The numbers of samples are 10,890. In this simulation, 10,000 and 890 samples are randomly chosen for $T_r$ and $T_e$, respectively. MPA modeling is relatively more complicated than LPF because of the large training data and inputs. Thus, MPA needs a robust training algorithm. Therefore, AdaNAQ and AdaMoQ are compared in this simulation. A NN with two hidden layers with 50 and 150 hidden neurons was chosen because a NN with one hidden layer could not obtain a small training error even if the hidden layer had 1,000 or more hidden neurons. Thus, the structure of the NN is 4-50-150-2. The results are shown in Table IV. The table shows that both algorithms can obtain small training and testing errors with a high convergence rate. Also, the number of iteration was almost the same. This can be seen from Fig. 8 which shows the error vs. iteration count for the minimum training error of 10 trials in the NN trained by AdaNAQ and AdaMoQ. On the other hand, AdaMoQ is faster in computation time, as in Fig. 9 showing the graph of training error vs. time. Therefore, the proposed AdaMoQ is faster than AdaNAQ. As a result, it is confirmed that the proposed algorithm is effective for large-scale and much complicated problems such as MPA.

### 4.2 Classification problems

In the function approximation problems, the error function defined was MSE. Here, the effectiveness of the proposed MoQ is shown for a different type of error function. Two classification benchmark problems [25, 28] are considered. In these problems, the error function $E(w)$ used is the Cross Entropy.
(CE) error as shown in (31),
\[
\min_{w \in \mathbb{R}^n} E(w) = \frac{1}{|T_r|} \sum_{p \in T_r} E_p(w),
\]
where,
\[
E_p(w) = -\sum_{l=1}^{L} d_{p,l} \log(o_{p,l}),
\tag{31}
\]
where \(d_{p,l}\) and \(o_{p,l}\) denote the \(l^{th}\) elements of \(d_p\) and \(o_p\), respectively, and \(L\) is the number of output units and denotes the number of classes. The hidden layer uses the sigmoid function while the output layer uses the softmax activation function given as
\[
\text{softmax}(z_l) = \frac{\exp(z_l)}{\sum_{l=1}^{L} \exp(z_l)}.
\]
From the simulations on examples in the previous section, it was observed that the first-order methods could not converge within the maximum number of iterations. However, in the case of classification problems used here, first-order methods, in particular Adam, have shown to be effective in terms of high classification accuracy and low cross entropy error. Thus in this section, we evaluate the performance of the proposed MoQ in comparison with Adam, QN, MSQN and NAQ. The hyperparameters of Adam are set to the default value as in [13]. The momentum coefficients \(\mu_k\) used in NAQ and MoQ are set to the adaptive values as in (12). The two classification benchmark problems used here are \(8 \times 8\) MNIST handwritten digits [25] and three-spirals [28] classification problems. Each trained neural network is evaluated by the median, average, best and worst accuracy, the average computation time (sec), and the average iteration counts (\(k\)) over 10 trials. The training accuracy is denoted by \(\text{Accuracy}_{\text{train}}\). The testing accuracy denoted by \(\text{Accuracy}_{\text{test}}\) is calculated using a test data set \(T_e\) which is independent of the training data set \(T_r\). The termination condition is experimentally set to \(\epsilon = 1.0 \times 10^{-6}\) and \(k_{\text{max}} = 150,000\).

4.2.1 8 \times 8 MNIST handwritten digit dataset
The first classification problem to evaluate the performance of each algorithm is the \(8 \times 8\) MNIST handwritten digit dataset [25–27], illustrated in Fig. 10. This dataset is the reduced version of the \(28 \times 28\) MNIST dataset [6] with 1797 samples which are divided into 75% (1347 samples) for \(T_r\) and 25% (450 samples) for \(T_e\), respectively [26]. The inputs and outputs of NN are 64 and 10 which are the numbers of the input data dimension and classes, respectively. The structure of the neural network is 64-10-10, that is, NN has a hidden layer with 10 neurons. The results are shown in Table V. From the table, it can be seen that Adam has converged with 100% whereas it could not converge within the maximum number of iteration counts in the function approximation problems. The median of training accuracy of Adam is 100% which is the highest among all algorithms. Although the testing error of Adam was almost the same as the other algorithms, Adam required more training time and iteration counts than the second-order methods. In the comparison of second-order methods, AdaNAQ and
Fig. 10. Examples of 8 × 8 MNIST handwritten digit dataset.

Table V. Summary of simulation results of 8 × 8 MNIST.

| Algorithm | $\mu_k$ | $\text{Accuracy}_{train}$(%) | $\text{Time}$ (sec) | Iteration counts (k) | $\text{Accuracy}_{test}$(%) | Convergence rate (%) |
|-----------|---------|-------------------------------|---------------------|-----------------------|-----------------------------|---------------------|
| Adam      | -       | 100 / 99.97 / 100 / 99.92    | 14                  | 5,481                 | 95.88 / 95.80 / 96.44 / 95.11 | 100                 |
| QN        | -       | 99.92 / 99.90 / 99.92 / 99.77 | 4.03                | 1,165                 | 96.22 / 96.13 / 96.88 / 95.55 | 100                 |
| MSQN      | -       | 99.92 / 99.91 / 99.92 / 99.77 | 5.13                | 1,211                 | 96.11 / 96.15 / 97.11 / 95.55 | 100                 |
| AdaNAQ    | -       | 99.70 / 99.70 / 99.85 / 99.48 | 0.46                | 73                    | 95.22 / 95.13 / 95.77 / 93.77 | 100                 |
| AdaMoQ    | -       | 99.70 / 99.72 / 99.85 / 99.62 | 0.27                | 72                    | 95.33 / 95.11 / 96.00 / 93.33 | 100                 |

Fig. 11. The training error vs. iteration counts of MNIST.

Fig. 12. The training error vs. time of MNIST.

AdaMoQ were able to obtain training and test accuracy close to QN and MSQN. However, QN and MSQN required more iteration counts and training time compared to AdaNAQ and AdaMoQ. The proposed AdaMoQ is the fastest among all algorithms. These results were more clear from Figs. 11–14 which show the graphs of training error vs. iteration counts, training error vs. time, test accuracy vs. iteration counts and test accuracy vs. time, respectively. Note that, in order to show a
clear graph, the $y$– and $x$–axes of Figs. 11 and 12 were set to logarithmic and linear, respectively. On the other hand, in Figs. 13 and 14, the $y$– and $x$–axes are linear and logarithmic, respectively. From Figs. 11 and 13, it can be observed that AdaNAQ and AdaMoQ converge in the early steps of iteration counts and they have almost the same performance. From Figs. 12 and 14, it is clear that the proposed AdaMoQ is faster than AdaNAQ. This means that the approximation of (14) in MoQ is also effective for cross entropy error function.

4.2.2 Three-spirals dataset
The final classification problem is the three spirals classification problem [28] illustrated in Fig. 15. This problem is known to be a relatively complex classification benchmark in NN training. The inputs of the network are $x$ and $y$ coordinates and the outputs are the numbers of classes $L = 3$. Therefore,
the network structure is 2-10-3. Each $L$ have 350 points and $|T_r|$ is 1050. The summary of training results is shown in Table VI. All algorithms can obtain almost the same training errors with 100% convergence. From the table, it can be seen that all algorithms have the same accuracies in this problem. Similar to the previous results, Adam required more iteration counts and time compared to QN based algorithms. Among the QN based algorithms, AdaMoQ converged fastest and with the least number of iterations. This result was clear from Figs. 16-19 which show the graphs of the training error vs. iteration counts, training error vs. time, training accuracy vs. iteration counts and training accuracy vs. time, respectively. Note that, same as the previous problem, the $y$– and $x$–axes are set to logarithmic and linear in Figs. 16 and 17, and Figs. 18 and 19, are linear and logarithmic, respectively. From Figs. 16 and 18, we can see that AdaMoQ converges with almost the same performance as that of AdaNAQ. In Figs. 17 and 19, it is clear that AdaMoQ is faster than AdaNAQ in terms of time. To validate the accuracies of the model trained with Adam and AdaMoQ, the trained models were evaluated on the test data. The testing results are shown in Figs. 20 and 21. From these figures, it can be seen that both of the train NNs can classify 2D planes well without overfitting.
Fig. 18. The training accuracy vs. iteration counts of three-spirals.

Fig. 19. The training accuracy vs. time of three-spirals.

Fig. 20. The test result of three-spirals trained by Adam.

Fig. 21. The test result of three-spirals trained by AdaMoQ.
5. Conclusion

This paper extends our study on our proposed Momentum quasi-Newton method (MoQ). The proposed method was an improvement of Nesterov’s accelerated quasi-Newton method (NAQ) so that Nesterov’s accelerated gradient was approximated as a weighted linear combination of the normal gradients by assuming the error function to be a quadratic function. The performance of the proposed method has been demonstrated through the training for two types of problems, such as the high nonlinearity function approximation and the classification problems. The proposed MoQ showed its effectiveness in all problems in this paper. The adaptive scheme of the momentum coefficients \( \mu_k \) was applied to the proposed MoQ for stable convergence. As a result, it has been confirmed that the proposed AdaMoQ is the efficient and practical training algorithm for NN compared to both the first and second-order conventional methods.

In the future, the adaptive scheme of the momentum parameters \( \mu_k \) will be analytically studied. The convergence property of the momentum acceleration of the quasi-Newton method will be clarified through this analysis. Furthermore, the implementation of stochastic (mini-batch) strategies \cite{27} and limited-memory schemes \cite{33} will be studied to demonstrate the effectiveness of the proposed algorithms for larger and deeper NN training for problems with high nonlinearities.

Acknowledgments

The author thanks Prof. Q.J. Zhang at Carleton University, Canada, for his support on the microwave circuit models. This work was supported by the Japan Society for the Promotion of Science (JSPS), KAKENHI (17K00350 and 20K11979).

References

\[1\] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning, MIT Press, 2016.
\[2\] S. Haykin, Neural Networks and Learning Machines 3rd, Pearson, 2009.
\[3\] H. Kabir, L. Zhang, M. Yu, P.H. Aaen, J. Wood, and Q.J. Zhang, “Smart modeling of microwave devices,” IEEE Microwave Magazine, vol. 11, no. 3, pp. 105–118, April 2010. DOI: 10.1109/MMM.2010.936079
\[4\] Q.J. Zhang, K.C. Gupta, and V.K. Devabhaktuni, “Artificial neural networks for RF and microwave design - from theory to practice,” IEEE Transactions on Microwave Theory and Techniques, vol. 51, no. 4, pp. 1339–1350, April 2003. DOI: 10.1109/TMTT.2003.809179
\[5\] H. Ninomiya, S. Wan, H. Kabir, Z. Zhang, and Q.J. Zhang, “Robust training of microwave neural network models using combined global/local optimization techniques,” IEEE MTT-S International Microwave Symposium (IMS) Digest, pp. 995–998, September 2008. DOI: 10.1109/MWSYM.2008.4633002
\[6\] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” Proc. IEEE, vol. 86, no. 11, pp. 2278–2324, November 1998. DOI: 10.1109/5.726791
\[7\] A. Kao and S.R. Poteet, (Eds.), Natural Language Processing and Text Mining, Springer Science & Business Media, 2007.
\[8\] J. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization,” The Journal of Machine Learning Research, vol. 12, no. 7, pp. 2121–2159, July 2011.
\[9\] M.D. Zeiler, “ADADELTA: An adaptive learning rate method,” arXiv print arXiv:1212.5701, December 2012.
\[10\] T. Tieleman and G. Hinton, “Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude,” COURSERA: Neural Networks for Machine Learning, vol. 4, no. 2, pp. 26–31, 2012.
\[11\] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, “On the importance of initialization and momentum in deep learning,” Proc. 30th International Conference on Machine Learning (ICML’13), vol. 28, pp. 1139–1147, June 2013. DOI: 10.5555/3042817.3043064
[12] Y. Nesterov, *Introductory Lectures on Convex Optimization: A Basic Course*, Springer Science & Business Media, vol. 87, 2003.

[13] D.P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” *Proc. 3rd International Conference for Learning Representations (ICLR’15)*, pp. 1–13, May 2015.

[14] H. Ninomiya, “A novel quasi-Newton optimization for neural network training incorporating Nesterov’s accelerated gradient,” *NOLTA*, vol. E8–N, no. 4, pp. 289–301, October 2017. DOI: 10.1587/nolta.8.289

[15] J. Nocedal and S.J. Wright, *Numerical Optimization Second Edition*, Springer Science & Business Media, 2006.

[16] S. Mahboubi, S. Indrapriyadarsini, H. Ninomiya, and H. Asai, “Momentum acceleration of quasi-Newton Training for Neural Networks,” *Proc. 16th Pacific Rim International Conference on Artificial Intelligence (PRICAI’19)*, Springer, Cham, vol. 11671, pp. 268–281, August 2019. DOI: 10.1007/978-3-030-29911-8_21

[17] W. Forst and D. Hoffmann, *Optimization - Theory and Practice*, Springer Science & Business Media, 2010.

[18] S. Mahboubi and H. Ninomiya, “A novel quasi-Newton with momentum training for microwave circuit models using neural networks,” *Proc. 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS’18)*, pp. 629–632, December 2018. DOI: 10.1109/ICECS.2018.8617883

[19] S. Mahboubi, S. Indrapriyadarsini, H. Ninomiya, and H. Asai, “A robust quasi-Newton training with adaptive momentum for microwave circuit models in neural networks,” *Journal of Signal Processing*, vol. 24, no. 1, pp. 11–17, January 2020. DOI: 10.2299/jsp.24.11

[20] J.A. Ford and I.A.R. Moghrabi, “Alternative parameter choices for multi-step quasi-Newton methods,” *Optimization Methods and Software*, vol. 2, no. 3–4, pp. 357–370, April 1993. DOI: 10.1080/10556789308805550

[21] I.A.R. Moghrabi, “Curvature-based quasi-Newton methods for optimization,” *International Journal of Pure and Applied Mathematics (IJPAM)*, vol. 119, no. 1, pp. 131–143, June 2018. DOI: 10.12732/ijpam.v119i1.11

[22] D.H. Li and H. Fukushima, “A modified BFGS method and its global convergence in nonconvex minimization,” *Journal of Computational and Applied Mathematics*, vol. 129, no. 1–2, pp. 15–35, April 2001. DOI: 10.1016/S0377-0427(00)00540-9

[23] S. Indrapriyadarsini, S. Mahboubi, H. Ninomiya, and H. Asai, “Implementation of a Modified Nesterov’s Accelerated Quasi-Newton Method on Tensorflow,” *Proc. 17th IEEE International Conference on Machine Learning and Applications (ICMLA’18)*, pp. 1147–1154, December 2018. DOI: 10.1109/ICMLA.2018.00185

[24] Y. Cao, G. Wang, and Q.J. Zhang, “A new training approach for parametric modeling of microwave passive components using combined neural networks and transfer functions,” *IEEE Transactions on Microwave Theory and Techniques*, vol. 57, no. 11, pp. 2727–2742, October 2009. DOI: 10.1109/TMTT.2009.2032476

[25] E. Alpaydin and C. Kaynak, “Optical recognition of handwritten digits data set,” *UCI Machine Learning Repository*, July 1998.

[26] D.P. Kroese, Z. Botev, T. Taimre, and R. Vaisman, *Data Science and Machine Learning: Mathematical and Statistical Methods*, CRC Press, 2019.

[27] S. Indrapriyadarsini, S. Mahboubi, H. Ninomiya, and H. Asai, “A stochastic quasi-newton method with Nesterov’s accelerated gradient,” *Proc. Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD’19)*, Springer, Cham, vol. 11906, pp. 743–760, September 2019. DOI: 10.1007/978-3-030-46150-8_43

[28] J. Bassey, X. Li, and L. Qian, “An experimental study of multi-layer multi-valued neural network,” *Proc. 2nd IEEE International Conference on Data Intelligence and Security (ICDIS’19)*, pp. 233–236, June 2019. DOI: 10.1109/ICDIS.2019.00043

[29] H. Ninomiya, “Dynamic sample size selection based quasi-Newton training for highly nonlinear function approximation using multilayer neural networks,” *Proc. IEEE International Joint
H. Ninomiya, “An improved online quasi-Newton method for robust training and its application to microwave neural network models,” Proc. IEEE International Joint Conference on Neural Networks (IJCNN’10), pp. 1–8, July 2010. DOI: 10.1109/IJCNN.2010.5596655

Sonnet, Sonnet Software, Inc, http://www.sonnetsoftware.com. (Last accessed March 5, 2021).

S. Ruder, “An overview of gradient descent optimization algorithm,” arXiv preprint arXiv:1609.04747, September 2016.

S. Mahboubi and H. Ninomiya, “A novel training algorithm based on limited-memory quasi-Newton method with Nesterov’s accelerated gradient in neural networks and its application to highly-nonlinear modeling of microwave circuit,” IARIA International Journal on Advances in Software, vol. 11, no. 3&4, pp. 323–334, November 2018.