# Quasi-Adam: Accelerating Adam using quasi-Newton approximations

Aditya Ranganath
Lawrence Livermore
National Laboratory
Livermore, CA, U.S.
Email: ranganath2@llnl.gov

Irabiel Romero Ruiz
Applied Mathematics
University of California, Merced
Merced, CA, U.S.
iromeroruiz@ucmerced.edu

Mukesh Singhal
Electrical Engineering
and Computer Science
Merced, CA, U.S.
msinghal@ucmerced.edu

Roummel Marcia
Applied Mathematics
University of California, Merced
Merced, CA, U.S.
rmarcia@ucmerced.edu

***Abstract*—Adam is arguably one of the most commonly used approach in deep learning and machine learning. With good regret bounds and empirical convergence proofs, the approach has produced many state-of-the-art models over a variety of problems. However, the method only uses gradient information at each iterate in addition to some moving averaged gradients and its corresponding moments from the past. In this paper, we propose a method that builds upon Adam and incorporates quasi-Newton matrices for approximating second derivatives. These Hessian approximations satisfy the so-called secant equation, which is the first-order Taylor series expansion of the gradient along the direction of the change in iterates. Judicious choices of quasi-Newton matrices can lead to guaranteed descent in the objective function and improved convergence. In this work, we integrate search directions obtained from using these quasi-Newton Hessian approximations with the Adam optimization algorithm. We provide convergence guarantees and demonstrate improved performance through an extensive experimentation on a variety of applications.**

***Index Terms*—Machine learning, deep learning, optimization, image-processing**

## I. Introduction

Stochastic gradient-based optimization plays a vital role in deep learning and machine learning. Most deep learning problems are cast as an optimization problem as follows:

$$\min_{\Theta} \sum_{i=1}^{N} f(\mathbf{x}_i, \mathbf{y}_i; \Theta), \tag{1}$$

where $f\colon \mathbb{R}^n \to \mathbb{R}$ is a nonlinear and nonconvex function, $\Theta \in \mathbb{R}^n$ is the vector of parameters of the neural network, $\mathbf{x}_i$ is the input observation and $\mathbf{y}_i$ is the corresponding label.

Gradient-based optimization methods are one of the most commonly used approaches in deep learning. This is owing to their fast computational nature (computing the gradient is asymtotically equal to a forward pass through the neural network) and stochastic nature (the gradients are evaluated only at certain input points). Due to these factors, it was able to propel research advances in the intersection of deep learning and optimization (see [1]). In recent years, a mutlitude of exponentially moving average and moment approaches have been proposed to optimize neural networks (see e.g. [2], [3], [4], [5]). The main objective of exponentially moving average is to limit the reliance of the update on the past gradient information instead of recent gradient information.

Quasi-Newton approximations, on the other hand, explicitly use information from the past (steps and change in gradients) to build an approximation of the Hessian. This approximation induces the curvature information using the secant information. We discuss this in detail in Sec. III. However, computing a step using quasi-Newton updates can be expensive due to its size and operations required. To overcome this, a limited-memory approach is generally used. Limited-memory BFGS (L-BFGS) is a very common approach where the Hessian approximation always stays positive-definite. We discuss this further in Sec. III. In recent work, quasi-Newton approaches have proven to be more deep learning friendly (see [6]).

In this paper, we propose *quasi-Adam*, a combination of an exponentially moving average and moment approach with the L-BFGS quasi-Newton update. The paper is divided into the following sections: In Sec. II, we discuss exponential moving average methods such as Adam and AdaGrad. In Sec. III, we discuss the quasi-Newton approaches, and their compact-representations. In Sec. IV, we provide the pseudo-code of the proposed approach and discuss the space and time-complexity of it. In Sec. VI, we discuss the experimental setup, testbed, datasets we will be using and models used for each dataset and the results of these experiments. In Sec. VII, we discuss the results obtained in Sec. VI and provide explanation and hypotheses based on the results. In Sec. VIII, we finally provide our concluding statements.

**Notation.** We denote the gradient of $f(\Theta)$ in (1) at the $t^{\text{th}}$ iteration by $\mathbf{g}_t = \nabla f(\Theta_t)$ and the Hessian approximation by $\mathbf{B}_t \approx \nabla^2 f(\Theta_t) \in \mathbb{R}^{n\times n}$. We denote the exact inverse of $\mathbf{B}_t$ by $\mathbf{H}_t$, i.e., $\mathbf{H}_t = \mathbf{B}_t^{-1}$. The learning rate is denoted by $\alpha_t$, and the scalar $g_t^2$ corresponds to $g_t^2 = \|\mathbf{g}_t\|_2^2$. The matrix $\mathbf{I}$ is the $n \times n$ identity matrix.

## II. Exponential moving average methods

Given a loss function $f(\Theta)$, gradient-based optimization approaches generate a sequence of iterates $\{\Theta_t\}$ that are computed using the following update:

$$\Theta_{t+1} = \Theta_t + \alpha_t \mathbf{p}_t,$$

where $\alpha_t$ is the learning rate and $\mathbf{p}_t$ is the search direction. For gradient-descent methods, $\mathbf{p}_t = -\mathbf{g}_t$. For highly nonlinear and nonconvex functions, such as the typical neural network loss function, large learning rate (or step sizes) do not guarantee a reduction in the loss function, causing a non-monotone behavior within the loss function. Likewise, a small learning rate leads to slow convergence. To tackle this problem, the concept of momentum and moment was introduced.

Momentum is the process of weighted averaging the gradients $\mathbf{g}_t$ over time $t$. Given the initial momentum vector $\mathbf{m}_0$ be initialized as a vector of zeros, the expression for $\mathbf{m}_t$ can be written as

$$\mathbf{m}_t = \Psi(\mathbf{g}_t, \mathbf{m}_{t-1}),$$

where $t \in [1, T]$ and $\Psi$ computes the weighted average between $\mathbf{g}_t$ and $\mathbf{m}_{t-1}$. This was first introduced by [7] and adapted by [4] and [Sutskever, unpublished 2012] in a deep learning setting, with some minor modifications. The motivation was to damp the non-monontone behaviour in regions of high curvature by averating over gradients with conflicting directions.

Hinton further improved upon the momentum based approach (see [8]) by employing a Root-Mean-Square (RMS) moving average which computes the weighted sum of $g_t^2$ over $t$. Given the initial moment $v_0 = 0$, the expression for $v_t$ can be written as

$$v_t = \Phi(g_t^2; v_{t-1}),$$

where $\Phi$ is a weighted sum of $v_{t-1}$ and $g_t^2$. The generalized expression for an exponential moving average update is written as

$$\mathbf{p}_t = -\frac{\Psi(\mathbf{g}_t, \mathbf{m}_{t-1})}{\Phi(g_t^2, v_t)}. \tag{2}$$

This moment term allows for the gradient to be normalized (in some way), helping the learning rate to work better. This propelled the use of momentum and moments in most mordern deep learning optimization algorithms.

The first major breakthrough was brought out by [2] who proposed Adam. The authors introduced an exponential decaying approach to the gradient update and the exponential moving average update employing and employed different weighted averages on the momentum and their moments.

Given $\beta_1, \beta_2 \in (0,1)$, the momentum vector $\mathbf{m}_t$ is defined as

$$\mathbf{m}_t = \frac{1}{(1-\beta_1^t)}\left(\beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\mathbf{g}_t\right), \tag{3}$$

and $v_t$ is defined as

$$v_t = \frac{1}{(1-\beta_2^t)}\left(\beta_2 v_{t-1} + (1-\beta_2) g_t^2\right). \tag{4}$$

Using (3) and (4), the Adam update is given by

$$\mathbf{p}_{\text{Adam}} = -\frac{\mathbf{m}_t}{\sqrt{v_t + \epsilon}}, \tag{5}$$

where $\epsilon = 10^{-8}$ is a scalar.

Thus, Adam uses an exponential moving average of the momentum and moments. Through the remainder of the paper, we explore the questions - Can we imporove upon an exponential moving average algorithm by inducing an approximation to the curvature information ? In Sec. III, we explore quasi-Newton appraoches to answer these questions.

## III. Quasi-Newton methods

Second-order approaches have the potential to exploit curvature information from second-order (Hessian) matrices. The iterate updates are defined using

$$\mathbf{p}_t = -[\nabla^2 f(\Theta_t)]^{-1} \mathbf{g}_t.$$

We note that the Hessian matrix $[\nabla^2 f(\Theta_t)]^{-1}$ is $n \times n$, which is computationally infeasible to form when $n$ is very large. To tackle with the dimensionality problem, users generally resolve to a finite difference method (see [9]) or the Pearlmutter technique (see [10]). These methods can be used in conjunction with a trust-region type approach (see [11]), which safeguards the step size, and a conjugate-gradient method which requires only *Hessian-vector* products without explicitly forming the Hessian. However, using the true Hessian can give rise to other issues, such as matrix singularity and non-positive definiteness.

Quasi-Newton approaches, on the other hand, only use approximations to the Hessian, which satisfy the *secant equation* given by

$$\mathbf{y}_{t-1} = \mathbf{B}_t \mathbf{s}_{t-1}, \tag{6}$$

where

$$\mathbf{y}_{t-1} = \nabla f(\Theta_t) - \nabla f(\Theta_{t-1}) \quad \text{and} \quad \mathbf{s}_{t-1} = \Theta_t - \Theta_{t-1}.$$

The L-BFGS method (see [12]) is one of the most commonly used quasi-Newton updates for $\mathbf{B}_t$ due to the guaranteed positive-definiteness of $\mathbf{B}_t$. Since we only work with the inverse of the Hessian to find the direction of descent, we will be only working with $\mathbf{H}_t = \mathbf{B}_t^{-1}$.

The matrix $\mathbf{H}_t$ is recursively defined as

$$\mathbf{H}_t = \left(\mathbf{I} - \frac{\mathbf{s}_{t-1}\mathbf{y}_{t-1}^\top}{\mathbf{y}_{t-1}^\top \mathbf{s}_{t-1}}\right)\mathbf{H}_{t-1}\left(\mathbf{I} - \frac{\mathbf{y}_{t-1}\mathbf{s}_{t-1}^\top}{\mathbf{y}_{t-1}^\top \mathbf{s}_{t-1}}\right) + \frac{\mathbf{s}_{t-1}\mathbf{s}_{t-1}^\top}{\mathbf{y}_{t-1}^\top \mathbf{s}_{t-1}}, \tag{7}$$

with

$$\mathbf{H}_0 = \frac{\mathbf{y}_0^\top \mathbf{s}_0}{\mathbf{y}_0^\top \mathbf{y}_0}\mathbf{I}.$$

We observe here that $\mathbf{H}_t$ represents an $n \times n$ matrix, which can get computationally expensive to store. Hence, this matrix is never stored explicitly. Rather, we only store the steps $\mathbf{s}_{t-1}$ and the change in gradients $\mathbf{y}_{t-1}$. Since (7) is only a two-rank update, it can be written as

$$\mathbf{H}_t = \gamma_{t-1}\mathbf{I} + \mathbf{\Gamma}_{t-1}\mathbf{M}_{t-1}\mathbf{\Gamma}_{t-1}^\top, \tag{8}$$

**Algorithm 1** Quasi-Adam Method

```
Require: $\alpha, \beta_1, \beta_2 \in [0,1), f(\Theta), \Theta_0, m_0 \leftarrow 0, v_0 \leftarrow 0, t \leftarrow 0$
  while $\Theta_t \neq \Theta^*$ do
    Compute $\mathbf{p}_{\text{Adam}}$ step using (5).
    Compute $\mathbf{p}_{\text{L-BFGS}}$ step using (9).
    Update $\Theta_{t+1} \leftarrow \Theta_t - \alpha_t(\mathbf{p}_{\text{Adam}} + \mathbf{p}_{\text{L-BFGS}})$.
  end while
```

where $\gamma_{t-1} = \mathbf{y}_{t-1}^\top \mathbf{s}_{t-1} / \mathbf{y}_{t-1}^\top \mathbf{y}_{t-1}$,

$$\mathbf{\Gamma}_{t-1} = \begin{bmatrix} \mathbf{s}_{t-1} & \gamma_{t-1}\mathbf{y}_{t-1} \end{bmatrix},$$

and

$$\mathbf{M}_{t-1} = \begin{bmatrix} \rho_{t-1} + \gamma_{t-1}\rho_{t-1}^2 \|y_{t-1}\|_2^2 & -\rho_{t-1} \\ -\rho_{t-1} & 0 \end{bmatrix},$$

with $\rho_{t-1} = (\mathbf{s}_{t-1}^\top \mathbf{y}_{t-1})^{-1}$. The search direction computed using the L-BFGS update is given by

$$\mathbf{p}_{\text{L-BFGS}} = -\mathbf{H}_t \mathbf{g}_t. \tag{9}$$

The steps are only accepted when $\mathbf{H}_t$ is positive definite, which is imposed when $\mathbf{s}_{t-1}^\top \mathbf{y}_{t-1} > 0$. Thus the matrix is invertible and provides a direction of descent.

Recently, practical L-BFGS methods have been proposed in a deep learning setting (see [6], [13], [14]). However, it is a common problem that L-BFGS performs very poorly on a variety of stochastic problems because of the use of stochastic gradients.

## IV. Proposed approach

From Sec. II and Sec. III, we take motivation from both approaches and define our new update rule called **quasi-Adam**. We present the update step in Algorithm 1. The approach uses a combination of both directions $\mathbf{p}_{\text{Adam}}$ and $\mathbf{p}_{\text{L-BFGS}}$ to improve the current step. In particular, the update step is given by

$$\Theta_{t+1} = \Theta_t - \alpha_t \left( \frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{v}_t + \epsilon}} + \mathbf{H}_t \mathbf{g}_t \right). \tag{10}$$

For the proposed approach, we use a memory size of 1. In the following sections, we discuss the space and time complexity of the proposed approach. The space and time complexity is presented as a modification to Adam - we only discuss the additional overhead for the proposed approach to Adam.

**Space Complexity**: Since we are only using 1 memory from the past, the space complexity is limited to $\mathcal{O}(n)$. We consume $\mathcal{O}(n)$ memory for saving the previous iterates weights from the model and $\mathcal{O}(n)$ for saving the gradients from past iterates, which gives us $\mathcal{O}(2n) \approx \mathcal{O}(n)$ asymptotically.

**Time complexity**: We need to perform the matrix vector product $\mathbf{H}_t \mathbf{g}_t$ in (9). The matrix $\mathbf{\Gamma}_{t-1}^\top$ is $2 \times n$, which means $\mathbf{\Gamma}_{t-1}^\top \mathbf{g}_t$ requires $\mathcal{O}(2n)$ operations. Each element in $\mathbf{M}_{t-1}$ is a scalar, which means $\mathbf{M}_{t-1}$ is a $2 \times 2$ matrix. Thus $\mathbf{M}_{t-1}\mathbf{\Gamma}_{t-1}^\top \mathbf{g}_t$ can be computed in $\mathcal{O}(4 + 2n)$ operations. Finally $\mathbf{\Gamma}_{t-1}\mathbf{M}_{t-1}\mathbf{\Gamma}_{t-1}^\top \mathbf{g}_t$ can be computed in $\mathcal{O}(4 + 4n)$ operations.

## V. Convergence

We analyze the convergence of the proposed approach using the framework by [15]. Given an arbitrary sequence of convex functions $\mathcal{C} = \{c_1, c_2, \ldots, c_T\}$, the goal is to predict the parameter $\Theta_t$ by optimizing it over the previous function $c_{t-1}$. This process of identifying an optimal $\Theta_t$ over $c_{t-1}$ is defined as an *online* algorithm. If $\Theta_t$ is selected by an algorithm $A$, we define the cost incurred by the algorithm $A$ as

$$L_A(T) = \sum_{t=1}^{T} c_t(\Theta_t). \tag{11}$$

In order to define the *offline* algorithm, we define a feasible convex set $\mathcal{F}$.

*Definition 1:* A set $\mathcal{F} \subseteq \mathbb{R}^n$ is convex if for all $\Theta, \Theta' \in \mathcal{F}$, $r\Theta + (1-r)\Theta' \in \mathcal{F}$ for all $r \in [0,1]$.

When the information on $\mathcal{C}$ and the convex subset $\mathcal{F}$ is available, the process of identifying the optimal $\Theta \in F$ is defined as *offline* algorithm (often also described as a static feasible solution).

Now, we formally introduce and define the *regret* function $R_A(T)$.

*Definition 2:* Given an algorithm A and a convex programming problem $(\mathcal{F}, \mathcal{C})$, if $\{\Theta_1, \Theta_2 \ldots\}$ are vectors selected by algorithm A, then the cost of A until time $T$ is given by (11). The cost of a static feasible solution $\Theta \in \mathcal{F}$ until time T is given by

$$L_\Theta(T) = \sum_{t=1}^{T} c_t(\Theta).$$

The regret of an algorithm A until time T is defined as

$$R_A(T) = L_A(T) - \min_{\Theta \in \mathcal{F}} L_\Theta(T).$$

The goal is to prove that the average regret $R_A(T)/T$ approaches 0 as $T \to \infty$.

For this, we begin by expanding the proposed update in (10):

$$\Theta_{t+1} = \Theta_t - \frac{\alpha_t}{1 - \beta_1^t} \left( \frac{\beta_{1,t}}{\sqrt{\hat{v}_t}} \hat{\mathbf{m}}_{t-1} - \frac{\beta_{1,t}}{\sqrt{\hat{v}_t}} \mathbf{g}_t + \mathbf{H}\mathbf{g}_t \right),$$

where $\beta_{1,t} = \beta_1 \lambda^{t-1}, \lambda \in (0,1)$. Here, $\hat{v}_t$ is the exponential moving average, defined as

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t},$$

with $\beta_2 \in (0,1]$, $\hat{m}_t$ is the bias-corrected first moment estimate defined as

$$\hat{\mathbf{m}}_t = \frac{\mathbf{m}_t}{1 - \beta_1^t},$$

where $\beta_1 \in (0,1]$, $m_t$ is the biased first moment estimate given by

$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1)\mathbf{g}_t,$$

$v_t$ is the biased second moment update given by

$$v_t = (1 - \beta_2) \sum_{i=1}^{t} \beta_2^{t-i} g_i^2,$$

We now make some mild assumptions for the iterates and their corresponding gradients.

*Assumption 1:* The distance between any $\Theta_t$ generated by the proposed approach is bounded. This means that $\|\Theta_m - \Theta_n\|_2 \leq D$.

*Assumption 2:* The gradients of function $f_t$ are bounded. This means, $\|\nabla f_t(\Theta)\|_2 \leq G$.

For the L-BFGS update, a step is acceptable if the condition $\mathbf{s}_t^\top \mathbf{y}_t > 0$ holds. We formally state this as a theorem below.

*Theorem 1:* For a convex set $\mathcal{F}$ and sequence of convex functions $\mathcal{C} = \{c_1, c_2, \ldots\}$, and for some step $\mathbf{s}_t = \Theta_{t+1} - \Theta_t$, where $\Theta_t, \Theta_{t+1} \in \mathcal{F}$, and change in gradient $\mathbf{y}_t = \nabla c_t(\Theta_{t+1}) - \nabla c_t(\Theta_t)$, where $c_t \in \mathcal{C}$ computed using a symmetric positive definite L-BFGS approximation, $\mathbf{s}_t$ and $\mathbf{y}_t$ will always satisfy the curvature condition $\mathbf{s}_t^\top \mathbf{y}_t > 0$.

In practice, the condition $\mathbf{s}_t^\top \mathbf{y}_t > 0$ (please refer Sec. III) is enforced by requiring $\mathbf{s}_t^\top \mathbf{y}_t \geq \varepsilon$ for some small $\varepsilon > 0$. It follows from Theorem 1 that $\|\mathbf{s}_t\|, \|\mathbf{y}_t\| \neq 0$ and that there exists some $c_l \in \mathbb{R}$ such that $0 < c_l \leq \mathbf{y}_t^\top \mathbf{y}_t$.

Given $\alpha_t = \alpha/\sqrt{t}$, we state the following theorem:

*Theorem 2:* Given Assumptions 1 and 2 hold, we get and upper bound on the regret as

$$\begin{aligned} R(T) \;\leq\; & \frac{D^2}{2\alpha(1-\beta_1)} \sum_{i=1}^{n} \sqrt{T\hat{v}_T} + \\ & \frac{(\beta_1+2)\alpha G}{(1-\beta_1)\sqrt{1-\beta_2}(1-\gamma)^2} \sum_{i=1}^{n} \|\mathbf{g}_{1:T,i}\| + \\ & \frac{n}{(1-\beta_1)(1-\lambda)^2}\Bigg[\frac{D^2G}{2\alpha} + \frac{D^2G}{2} + \\ & \qquad \frac{2D^2G^5}{c_l^2} + \frac{4\alpha D^2 G^5}{c_l^2}\Bigg]. \end{aligned}$$

From Theorem 2, the corollary follows:

*Corollary 1:* Quasi-Adam achieves the following guarantee:

$$R(T) = O(\frac{1}{\sqrt{T}}).$$

Thus, $\lim_{T\to\infty} R(T) \to 0$.

## VI. Experimental Setup

### A. Testbed

All the experimenst were conducted using PyTorch (see [16]) libraries using two NVIDIA 1080 Ti graphics cards over an Intel i7-7700K CPU. For Adam, we conducted a variety of experiments to choose the hyperparameter based on the applications. The experiments included different batch size $\{256, 512, 1,024, 2,048, 8,192\}$, with different learning rates $\{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 0.9\}$. We present results with the best hyperparameters for Adam and the proposed approach for each experiment. To understand the performance change between the proposed approach and Adam, we present the results where this change is most prominent. This is reflected in the first half of the training response. We observe that both the methods eventually converge to the same result after a large number of epochs.

### B. Models

We use three different types of networks - an MNIST classifier for classifying MNIST and Fashion-MNIST dataset, ResNet34 to classifiy images in the SVHN dataset and an Autoencoder for MNIST and Fashion-MNIST reconstruction.

**MNIST classifier:** We design a shallow neural network to classify the MNIST dataset. The model has two convolutional layers, and three fully-connected dense layers. Each convolutional layer is followed by a maxpooling layer and ReLU activation function. Fully connected layers are followed by a ReLU activation function.

**Resnet34**: Resnet34 ([17]) is a deep learning model with 34 blocks, which contain two convolutional layers with skip connections between blocks and ReLU activation layers in between. The network contains approximately 21.8 million parameters. The network was designed to train over the ImageNet dataset. Since we are using SVHN here, the network has been modified accordingly.

**Autoendcoder:** The encoder has a shallow 3 layer convolutional architecture and the decoder has a shallow 3 convolutional tranpose layer. The main purpose of the network is to reconstruct the images from its original image. The images are fed to the encoder, which compresses the image. This is then inflated/expanded by the decoder. The network has 87,125 parameters. We use the same network architecture for FMNIST reconstruction.

### C. Datasets

We use three different datasets for two types of tasks - MNIST, FMNIST and SVHN.

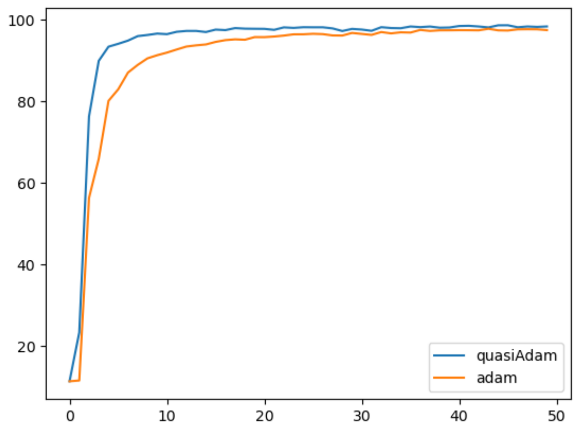
**MNIST:** MNIST is a dataset of $28 \times 28$ pixel handwritten digits from (0-9). These images are greyscaled single channel images with 5,000 training examples, 1,000 validation examples and 1,000 testing examples per class.

**Fashion-MNIST:** The Fashion-MNIST dataset (see [18]) consists of $28 \times 28$ pixel black-and-white images of clothing items such as shirt, automobile, shoes etc. The dataset has 10 classes and 6,000 images per class. This is further parted into 4,000 training examples, 2000 testing examples and 1,000 testing examples per class.

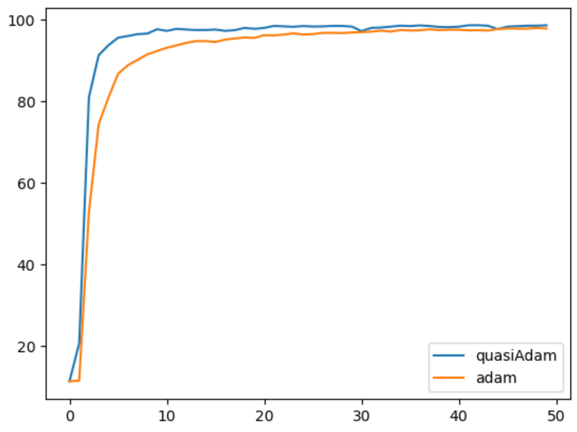
**Street View House Number:** The Street View House Number (SVHN) (see [19]) is a dataset with of street view images (numbers in addresses) extracted from Google Street View images. The dataset contains $32 \times 32$ RGB images of these street view images, cropped and separated into individual numbers ranging from (0-9). The training set contains 73,257 images while the testing set contains 26,032 images.

### D. Experiments

**Experiment I: MNIST image classification.** We use the MNIST classifier to classify the MNIST dataset and use cross-entropy as the loss function. We train the network with a batch-size of 256 and 512 images and a cross-entropy loss function. We use a learning rate of $1 \times 10^{-2}$ for Adam and quasi-Adam.

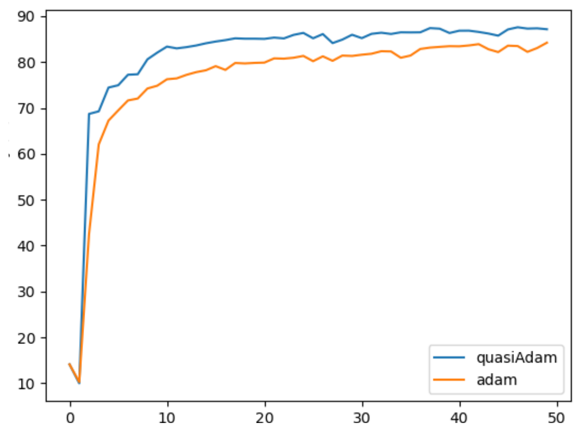


(a) Batch-size = 256

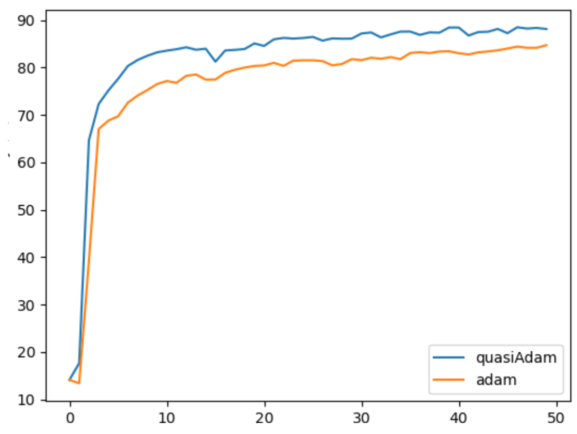


(b) Batch-size = 512

Fig. 1. **Experiment I:** MNIST image classification results for Adam and the proposed method, quasi-Adam. (a) Testing accuracy for batch-size of 256. (b) Testing accuracy for batch-size of 512. The $y$-axis represents the classification accuracy and the $x$-axis represents the batch-iteration. Note that for both batch-sizes, quasi-Adam outperforms Adam.



(a) Batch-size = 256



(b) Batch-size = 512

Fig. 2. **Experiment II:** Fashion-MNIST image classification results for Adam and the proposed method, quasi-Adam. (a) Testing accuracy for batch-size of 256. (b) Testing accuracy for batch-size of 512. The $y$-axis represents the classification accuracy and the $x$-axis represents the batch-iteration. Note that for both batch-sizes, quasi-Adam outperforms Adam.

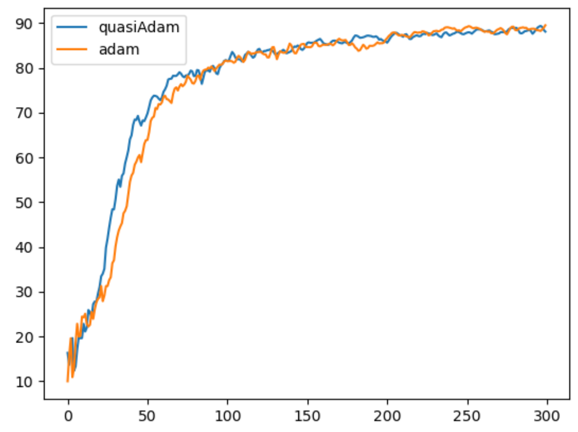
We present the testing response after each batch-iteration in Fig. 1.

**Experiment II: Fashion-MNIST image classification**. We use FashionMNIST dataset with the MNIST classifier. This is possible since the dimensions of the images are the same. We use a learning rate of $1 \times 10^{-2}$ for both Adam and quasi-Adam, and a cross-entropy loss function. We present the testing accuracy for each batch-training iteration with a size of 256 and 512 images in Fig. 2.

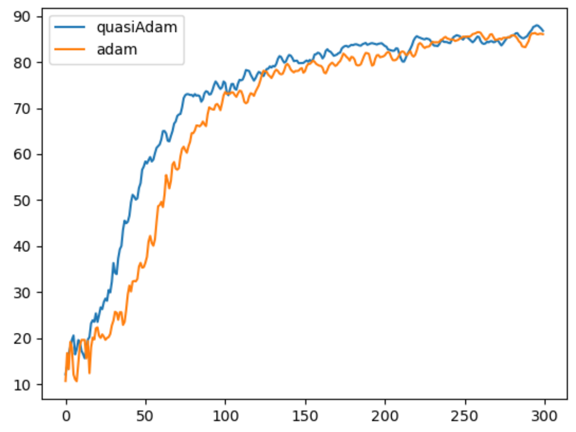
**Experiment III: SVHN image classification**. For this task, we use the ResNet34 neural network. We train with a batch size of 256 and 128 images. We use the negative log-likelihood loss function with a learning rate of $1 \times 10^{-2}$ for both Adam and quasi-Adam. The results presented in Fig. 3 show the testing accuracy after each batch-training iteration.

**Experiment IV: MNIST image reconstruction**. For this task, we use the autoencoder to reconstruct the images. We use a batch-size of 256 images and the Mean-Squared Error (MSE) loss function with a learning rate of $10^{-2}$ for Adam and quasi-Adam. We observe both the training loss and the testing loss for each of the tasks. The results in Fig. 4 show the testing accuracy after each batch-iteration.

**Experiment V: FMNIST image reconstruction**. For the autoencoder reconstruction experiment, we use the autoencoder from Experiment IV (the images are of the same dimensions). We use the same learning rate of $1 \times 10^{-3}$ for Adam and the

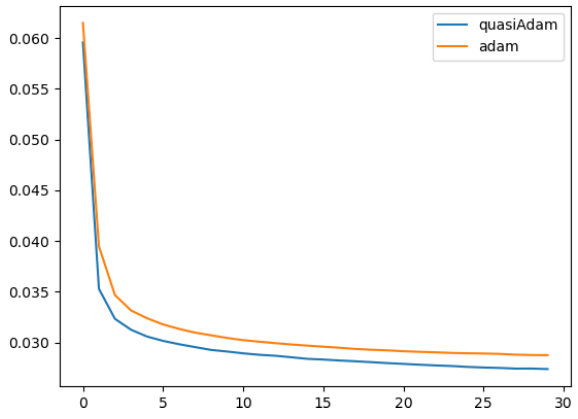


(a) Batch-size = 256

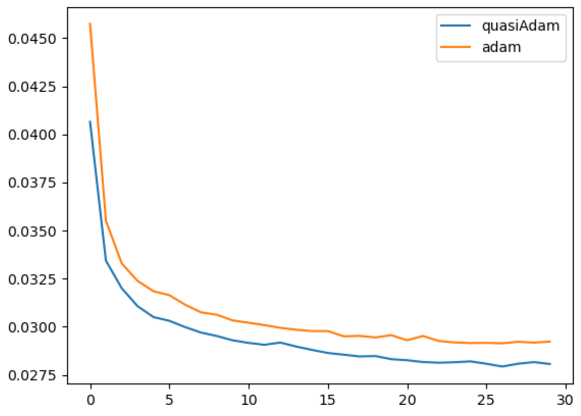


(b) Batch-size = 128

Fig. 3. **Experiment III:** SVHN image classification results for Adam and the proposed method, quasi-Adam. (a) Testing accuracy for batch-size of 256. (b) Testing accuracy for batch-size of 128. The $y$-axis represents the classification accuracy and the $x$-axis represents the batch-iteration. Note that for both batch-sizes, quasi-Adam outperforms Adam.



(a) Training loss



(b) Testing loss

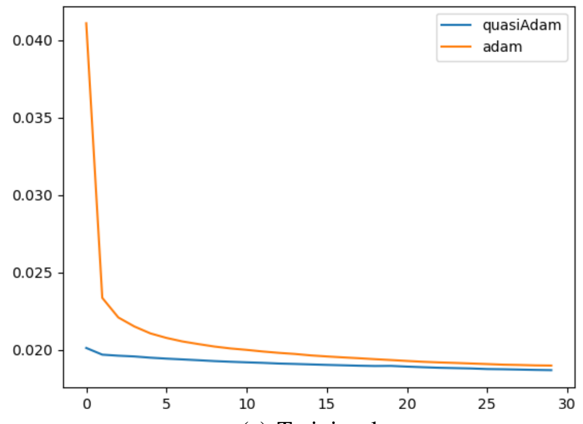
Fig. 4. **Experiment IV:** Autoencoder MNIST Reconstruction for Adam and the proposed method, quasi-Adam. (a) Training loss. (b) Testing response. The $x$-axis represents the number of epochs and $y$-axis represents the average mean-squared error loss for each epoch. Note that in both training and testing responses, quasi-Adam outperforms Adam.

proposed approach and the MSE loss function. We present the training and the testing response in Fig. 5.

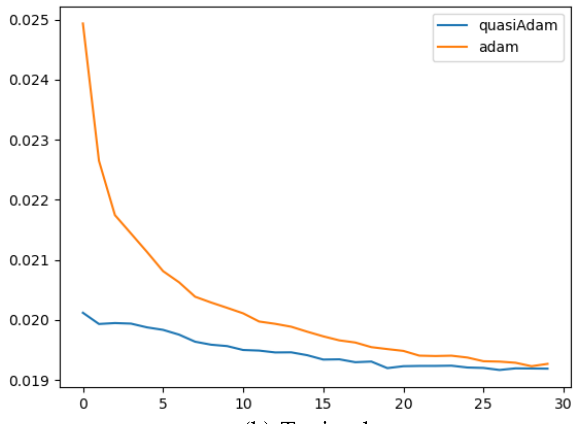
## VII. Discussion

The results from Sec. VI elucidates the performance improvement of quasi-Adam over Adam. This can be attributed to both participating approaches - Adam and L-BFGS. The Adam approach uses an exponential moving momentum $\mathbf{m}_t$ and a exponential moving moment $v_t$. This allows for the learning rate to work better at providing an adequate descent in the loss function in addition to treating the non-monotone behaviour of the loss function. However, it only exploits the gradient information, which is still a steepest descent direction. This may cause the iterates to dampen in a *plateau* region of the manifold, commonly referred to as saddle points. The L-BFGS approach, on the other hand, is able to provide a descent direction which exploits the curvature information using the secant equation. Since the Hessian information is induced in this matrix $\mathbf{H}_t$ in (9), the iterate potentially escapes this saddle point and addresses the sadle point problem more effectively. However, computing a large memory Hessian approximation can be computationally expensive.

The proposed approach was able to tackle both the problems - escaping saddle points and containing computational expense. This combination of methods yielded an improvement in training performance across a variety of applications. Thus, convergence can be *expedited* with a very small computational overhead.



(a) Training loss



(b) Testing loss

Fig. 5. **Experiment V:** Autoencoder FMNIST Reconstruction results for Adam and the proposed method, quasi-Adam. (a) The training loss. (b) Testing response. The $x$-axis represents the number of epochs and the $y$-axis represents the average mean-squared error loss for each epoch. Note that in both training and testing responses, quasi-Adam outperforms Adam.

## VIII. Conclusion

In this paper, we proposed a new algorithm which uses a quasi-Newton update in conjunction with a moment estimation update. We show that the curvature information from the quasi-Newton approach improve upon an exponential moving average method. Through thorough experimentation, we were able to show that the proposed approach was able to outperform Adam. We provided concrete convergence proofs and discuss the complexity analysis for space and time. We found quasi-Adam to be robust and suited across a variety of applications in the field of machine learning.

## Acknowledgment

R. Marcia's research is partially supported by NSF Grants DMS 1840265 and CCF 2343610. This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract LLNL-PROC-869166.

## References

[1] R. M. Schmidt, F. Schneider, and P. Hennig, "Descending through a crowded valley-benchmarking deep learning optimizers," in *International Conference on Machine Learning*, pp. 9367–9376, PMLR, 2021.
[2] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
[3] L. Luo, Y. Xiong, Y. Liu, and X. Sun, "Adaptive gradient methods with dynamic bound of learning rate," *arXiv preprint arXiv:1902.09843*, 2019.
[4] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization.," *Journal of machine learning research*, vol. 12, no. 7, 2011.
[5] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of adam and beyond," *arXiv preprint arXiv:1904.09237*, 2019.
[6] D. Goldfarb, Y. Ren, and A. Bahamou, "Practical quasi-newton methods for training deep neural networks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 2386–2396, 2020.
[7] Y. Nesterov, "A method for solving the convex programming problem with convergence rate $\mathcal{O}(1/k^2)$," *Proceedings of the USSR Academy of Sciences*, vol. 269, pp. 543–547, 1983.
[8] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent," *Cited on*, vol. 14, no. 8, p. 2, 2012.
[9] J. Martens *et al.*, "Deep learning via hessian-free optimization.," in *ICML*, vol. 27, pp. 735–742, 2010.
[10] B. A. Pearlmutter, "Fast exact multiplication by the hessian," *Neural computation*, vol. 6, no. 1, pp. 147–160, 1994.
[11] A. Ranganath, O. DeGuchy, M. Singhal, and R. F. Marcia, "Second-order trust-region optimization for data-limited inference," in *2021 29th European Signal Processing Conference (EUSIPCO)*, pp. 2059–2063, IEEE, 2021.
[12] D. C. Liu and J. Nocedal, "On the limited memory bfgs method for large scale optimization," *Mathematical programming*, vol. 45, no. 1-3, pp. 503–528, 1989.
[13] A. S. Berahas, M. Jahani, P. Richtárik, and M. Takáč, "Quasi-newton methods for machine learning: forget the past, just sample," *Optimization Methods and Software*, vol. 37, no. 5, pp. 1668–1704, 2022.
[14] X. Wang, S. Ma, D. Goldfarb, and W. Liu, "Stochastic quasi-newton methods for nonconvex stochastic optimization," *SIAM Journal on Optimization*, vol. 27, no. 2, pp. 927–956, 2017.
[15] M. Zinkevich, "Online convex programming and generalized infinitesimal gradient ascent," in *Proceedings of the 20th international conference on machine learning (icml-03)*, pp. 928–936, 2003.
[16] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, pp. 8024–8035, Curran Associates, Inc., 2019.
[17] 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*, pp. 770–778, 2016.
[18] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.
[19] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.