A block coordinate descent optimizer for classification problems exploiting convexity

Ravi G. Patel, Nathaniel A. Trask, Mamikon A. Gulian, Eric C. Cyr
Center for Computing Research, Sandia National Laboratories
Albuquerque, New Mexico, 87123
{rgpatel, natrask, mgulian, eccyr}@sandia.gov

Abstract

Second-order optimizers hold intriguing potential for deep learning, but suffer from increased cost and sensitivity to the non-convexity of the loss surface as compared to gradient-based approaches. We introduce a coordinate descent method to train deep neural networks for classification tasks that exploits global convexity of the cross-entropy loss in the weights of the linear layer. Our hybrid Newton/Gradient Descent (NGD) method is consistent with the interpretation of hidden layers as providing an adaptive basis and the linear layer as providing an optimal fit of the basis to data. By alternating between a second-order method to find globally optimal parameters for the linear layer and gradient descent to train the hidden layers, we ensure an optimal fit of the adaptive basis to data throughout training. The size of the Hessian in the second-order step scales only with the number weights in the linear layer and not the depth and width of the hidden layers; furthermore, the approach is applicable to arbitrary hidden layer architecture. Previous work applying this adaptive basis perspective to regression problems demonstrated significant improvements in accuracy at reduced training cost, and this work can be viewed as an extension of this approach to classification problems. We first prove that the resulting Hessian matrix is symmetric semi-definite, and that the Newton step realizes a global minimizer. By studying classification of manufactured two-dimensional point cloud data, we demonstrate both an improvement in validation error and a striking qualitative difference in the basis functions encoded in the hidden layer when trained using NGD. Application to image classification benchmarks for both dense and convolutional architectures reveals improved training accuracy, suggesting gains of second-order methods over gradient descent. A Tensorflow implementation of the algorithm is available at github.com/rgp62/.

A Newton/gradient coordinate descent optimizer for classification

Denote by $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N_{\text{data}}}$ data/label pairs, and consider the following class of deep learning architectures:

$$\mathcal{L}(\mathbf{W}, \xi, \mathcal{D}) = \sum_{(\mathbf{x}_i,\mathbf{y}_i)\in\mathcal{D}} \mathcal{L}_{\text{CE}}(\cdot; \mathbf{y}_i) \circ \mathcal{F}_{\text{SM}} \circ \mathcal{F}_{\text{LL}}(\cdot; \mathbf{W}) \circ \mathcal{F}_{\text{HL}}(\mathbf{x}_i; \xi), \quad (1)$$

where $\mathcal{L}_{\text{CE}}, \mathcal{F}_{\text{SM}}, \mathcal{F}_{\text{LL}}$ and $\mathcal{F}_{\text{HL}}$ denote a cross-entropy loss, softmax layer, linear layer, and hidden layer, respectively, and $\circ$ is composition. We denote linear layer weights by $\mathbf{W}$, and consider a general class of hidden layers (e.g. dense networks, convolutional networks, etc.), denoting associated weights and biases by the parameter ξ. The final three layers are expressed as

$$\begin{aligned} \mathcal{L}_{\text{CE}}(\mathbf{x}; \mathbf{y}) &= -\sum_{i=1}^{N_c} y^i \log x^i, \\ \mathcal{F}^i_{\text{SM}}(\mathbf{x}) &= \frac{\exp(x_i)}{\sum_{j=1}^{N_c} \exp(x_j)}, \\ \mathcal{F}^i_{\text{LL}}(\mathbf{x}) &= \mathbf{W}\mathbf{x}, \end{aligned} \quad (2)$$

and map $\mathcal{F}_{\text{HL}} : \mathbb{R}^{N_{\text{in}}} \to \mathbb{R}^{N_{\text{basis}}}$; $\mathcal{F}_{\text{LL}} : \mathbb{R}^{N_{\text{basis}}} \to \mathbb{R}^{N_{\text{classes}}}$; $\mathcal{F}_{\text{SM}} : \mathbb{R}^{N_{\text{classes}}} \to \mathbb{R}^{N_{\text{classes}}}$; and $\mathcal{L}_{\text{CE}} : \mathbb{R}^{N_{\text{classes}}} \to \mathbb{R}$. Here, N_{basis} is the dimension of the output of the hidden layer; this notation is explained in the next paragraph. The standard classification problem is to solve

$$(\mathbf{W}^*, \xi^*) = \underset{\mathbf{W},\xi}{\operatorname{argmin}}\, \mathcal{L}(\mathbf{W}, \xi, \mathcal{D}). \quad (3)$$

The recent work by Cyr et al. (2019) performed a similar partition of weights into linear layer weights $\mathbf{W}$ and hidden layer weights ξ for regression problems. Two important observations were made using this decomposition. First, the output of the hidden layers can be treated as an adaptive basis with the learned weights $\mathbf{W}$ corresponding to the coefficients producing the prediction. Second, holding ξ fixed leads to a linear least squares problem for the basis coefficients $\mathbf{W}$ that can be solved for a global minimum. This work builds on these two observations for classification problems. The output of the hidden layers $\mathcal{F}_{\text{HL}}$ defines a basis

$$\Phi_\alpha(\cdot, \xi) : \mathbb{R}^{N_{\text{in}}} \to \mathbb{R} \text{ for } \alpha = 1 \dots N_{\text{basis}} \quad (4)$$

where $\Phi_\alpha(x, \xi)$ is row α of $\mathcal{F}_{\text{HL}}(x, \xi)$. Thus the input to the softmax classification layer are N_{classes} functions, each defined using the adaptive basis Φ_α and a single row of the weight matrix $\mathbf{W}$. The crux of this approach to classification is the observation that for all ξ, the function

$$\mathcal{S}(\mathbf{W}, \mathcal{D}) = \mathcal{L}(\mathbf{W}, \xi, \mathcal{D}) \quad (5)$$

is convex with respect to $\mathbf{W}$, so the global minimizer

$$\mathbf{W}^* = \underset{\mathbf{W}}{\operatorname{argmin}}\, \mathcal{S}(\mathbf{W}, \mathcal{D}) \quad (6)$$

may be obtained via Newton iteration with line search. In the Algorithm section, we introduce a coordinate-descent optimizer that alternates between a globally optimal solution of (6) and a gradient descent step minimizing (3). Combining this with the interpretation of the hidden layer as providing a data-driven adaptive basis, this ensures that during training the parameters evolve along a manifold providing optimal fit of the adaptive basis to data (Cyr et al. 2019). We summarize this perspective in the section on Relation to previous works, and in the Results section we investigate how this approach differs from stochastic gradient descent (GD), both in accuracy and in the qualitative properties of the hidden layer basis.

Convexity analysis and invertibility of the Hessian

In what follows, we use basic properties of convex functions (Boyd, Boyd, and Vandenberghe 2004) and the Cauchy-Schwartz inequality (Folland 1999) to prove that $\mathcal{S}$ in (5) is convex. Recall that convexity is preserved under affine transformations. We first note that $\mathcal{L}_{\text{LL}}(\mathbf{W};\mathcal{D},\xi)$ is linear. By (1), it remains only to show that the composition $\mathcal{L}_{\text{CE}} \circ \mathcal{F}_{\text{SM}}$ is convex. We write, for any data vector $\mathbf{y}$,

$$\begin{aligned}
&\mathcal{L}_{\text{CE}} \circ \mathcal{F}_{\text{SM}}(\mathbf{x};\mathbf{y}) \\
&\quad = -\sum_{i=1}^{N_{\text{classes}}} y_i \log\left(\frac{\exp(x_i)}{\sum_{j=1}^{N_{\text{classes}}}\exp(x_j)}\right) \\
&\quad = -\sum_{i=1}^{N_{\text{classes}}} y_i x_i + \sum_{i=1}^{N_{\text{classes}}} y_i \log\left(\sum_{j=1}^{N_{\text{classes}}} \exp(x_j)\right) \\
&\quad = -\sum_{i=1}^{N_{\text{classes}}} y_i x_i + \log\left(\sum_{j=1}^{N_{\text{classes}}} \exp(x_j)\right).
\end{aligned}$$

The first term above is affine and thus convex. We prove the convexity of the second term $f(\mathbf{x}) := \log\left(\sum_{i=1}^{N_{\text{classes}}} \exp(x_i)\right)$ by writing

$$\begin{aligned}
&f(\theta\mathbf{x} + (1-\theta)\mathbf{y}) \\
&\quad = \log\left(\sum_{i=1}^{N_{\text{classes}}} (\exp(x_i))^{\theta} (\exp(y_i))^{1-\theta}\right).
\end{aligned}$$

Applying Cauchy-Schwartz with $1/p = \theta$ and $1/q = 1-\theta$,

$$\begin{aligned}
&f(\theta\mathbf{x} + (1-\theta)\mathbf{y}) \\
&\quad \leq \log\left(\left(\sum_{i=1}^{N_{\text{classes}}} \exp(x_i)\right)^{\theta} \left(\sum_{i=1}^{N_{\text{classes}}} \exp(y_i)\right)^{1-\theta}\right) \\
&\quad = \theta f(\mathbf{x}) + (1-\theta) f(\mathbf{y}).
\end{aligned}$$

Thus f, and therefore $\mathcal{L}_{\text{CE}} \circ \mathcal{F}_{\text{SM}}$ and $\mathcal{S}$, are convex. As a consequence, the Hessian H of $\mathcal{S}$ with respect to $\mathbf{W}$ is a symmetric positive semi-definite function, allowing application of a convex optimizer in the following section to realize a global minimum.

```
Data: batch $\mathcal{B} \subset \mathcal{D}, \xi_{\text{old}}, \mathbf{W}_{\text{old}}, \alpha, \rho$
Result: $\xi_{\text{new}}, \mathbf{W}_{\text{new}}$
for $j \in \{1, ..., \texttt{newton\_steps}\}$ do
    Compute gradient $G = \nabla_{\mathbf{W}} S(\mathbf{W}_{\text{old}}, \mathcal{B})$ and
      Hessian $H = \nabla_{\mathbf{W}} \nabla_{\mathbf{W}} S(\mathbf{W}_{\text{old}}, \mathcal{B})$ ;
    Solve $H\boldsymbol{s} = -G$;
    $\mathbf{W}^{\dagger} \leftarrow \mathbf{W}_{\text{old}} + \boldsymbol{s}$;
    $\lambda \leftarrow 1$;
    while $S(\mathbf{W}^{\dagger}, \mathcal{B}) > S(\mathbf{W}_{\text{old}}, \mathcal{B}) + \alpha\lambda G \cdot \boldsymbol{s}$ do
        $\lambda \leftarrow \lambda\rho$;
        $\mathbf{W}^{\dagger} \leftarrow \mathbf{W}_{\text{old}} + \lambda\boldsymbol{s}$;
    end
end
$\mathbf{W}_{\text{new}} \leftarrow \mathbf{W}^{\dagger}$;
$\xi_{\text{new}} \leftarrow \text{GD}(\xi_{\text{old}}, \mathcal{B}, \mathbf{W}_{\text{new}})$;
```

Algorithm 1: Application of coordinate descent algorithm for classification to a single batch $\mathcal{B} \subset \mathcal{D}$. For the purposes of this work, we use $\rho = 0.5$ and $\alpha = 10^{-4}$.

Algorithm

Application of the traditional Newton method to the problem (3) would require solution of a dense matrix problem of size equal to the total number of parameters in the network. In contrast, we alternate between applying Newton's method to solve only for $\mathbf{W}$ in (6) and a single step of a gradient-based optimizer for the remaining parameters ξ; the Newton step therefore scales with the number of weights ($N_{\text{basis}} \times N_{\text{classes}}$) in the linear layer. Since $\mathcal{S}$ is convex, Newton's method with appropriate backtracking or trust region may be expected to achieve a global minimizer. We pursue a simple backtracking approach, taking the step direction and size from standard Newton and repeatedly reducing the step direction until the Armijo condition is satisfied, ensuring an adequate reduction of the loss (Armijo 1966; Dennis Jr and Schnabel 1996). For the gradient descent step we apply Adam (Kingma and Ba 2014), although one may apply any gradient-based optimizer; we denote such an update to the hidden layers for fixed $\mathbf{W}$ by the function $\text{GD}(\xi, \mathcal{B}, \mathbf{W})$. To handle large data sets, stochastic gradient descent (GD) updates parameters using gradients computed over disjoint subsets $\mathcal{B} \subset \mathcal{D}$ (Bottou 2010). To expose the same parallelism, we apply our coordinate descent update over the same batches by solving (6) restricted to $\mathcal{B}$. Note that this implies an optimal choice of $\mathbf{W}$ over $\mathcal{B}$ only. We summarize the approach in Alg. 1.

While H and G may be computed analytically from (2), we used automatic differentiation for ease of implementation. The system $H\boldsymbol{s} = -G$ can be solved using either a dense or an iterative method. Having proven convexity of $\mathcal{S}$ in (6), and thus positive semi-definiteness of the Hessian, we may apply a conjugate gradient method. We observed that solving to a relatively tight residual resulted in overfitting during training, while running a fixed number N_{cg} of iterations improved validation accuracy. Thus, we treat N_{cg} as a hyperparameter in our studies below. We also experimented with dense solvers; due to rank deficiency we considered a pseudo-inverse of the form $H^\dagger = (H + \epsilon I)^{-1}$, where taking a finite $\epsilon > 0$ provided similar accuracy gains. We speculate that these approaches may be implicitly regularizing the training. For brevity we only present results using the iterative approach; the resulting accuracy was comparable to the dense solver. In the following section we typically use only a handful of Newton and CG iterations, so the additional cost is relatively small.

We later provide convergence studies comparing our technique to GD using the Adam optimizer and identical batching. We note that a lack of optimized software prevents a straightforward comparison of the performance of our approach vs. standard GD; while optimized GPU implementations are already available for GD, it is an open question how to most efficiently parallelize the current approach. For this reason we compare performance in terms of iterations, deferring wall-clock benchmarking to a future work when a fair comparison is possible.

Relation to previous works

We seek an extension of Cyr et al. (2019). This work used an adaptive basis perspective to motivate a block coordinate descent approach utilizing a linear least squares solver. The training strategy they develop can be found under the name of variable projection, and was used to train small networks (McLoone et al. 1998; Pereyra, Scherer, and Wong 2006). In addition to the work in Cyr et al. (2019), the perspective of neural networks producing an adaptive basis has been considered by several approximation theorists to study the accuracy of deep networks (Yarotsky 2017; Opschoor, Petersen, and Schwab 2019; Daubechies et al. 2019). The combination of the adaptive basis perspective combined with the block coordinate descent optimization demonstrated dramatic increases in accuracy and performance in Cyr et al. (2019), but was limited to an ℓ_2 loss. None of the previous works have considered the generalization of this approach to training deep neural networks with a cross-entropy loss typically used in classification as we develop here.

Bottou, Curtis, and Nocedal (2018) provides a mathematical summary on the breadth of work on numerical optimizers used in machine learning. Several recent works have sought different means to incorporate second-order optimizers to accelerate training and avoid issues with selecting hyperparameters and training schedules (Osawa et al. 2019, 2020; Botev, Ritter, and Barber 2017; Martens 2010). Some pursue a quasi-Newton approach, defining approximate Hessians, or apply factorization to reduce the effective bandwidth of the Hessian (Botev, Ritter, and Barber 2017; Xu, Roosta, and Mahoney 2019). Our work pursues a (block) coordinate descent strategy, partitioning degrees of freedom into sub-problems amenable to more sophisticated optimization (Nesterov 2012; Wright 2015; Blondel, Seki, and Uehara 2013). Many works have successfully employed such schemes in ML contexts (e.g. (Blondel, Seki, and Uehara 2013; Fu 1998; Shevade and Keerthi 2003; Clarkson, Hazan, and Woodruff 2012)), but they typically rely on stochastic partitioning of variables rather than the partition of the weights of deep neural networks into hidden layer variables and their complement pursued here. The strategy of extracting convex approximations to nonlinear loss functions is classical (Bubeck 2014), and some works have attempted to minimize general loss functions by minimizing surrogate ℓ_2 problems (Barratt and Boyd 2020).

Results

We study the performance and properties of the NGD algorithm as compared to the standard stochastic gradient descent (GD) on several benchmark problems with various architectures. We start by applying dense network architectures to classification in the peaks problem. This allows us to plot and compare the qualitative properties of the basis functions $\Phi_\alpha(\cdot, \xi)$ encoded in the hidden layer (4) when trained with the two methods. We then compare the performance of NGD and GD for the standard image classification benchmarks CIFAR-10, MNIST, and Fashion MNIST using both dense and convolutional (ConvNet) architectures. Throughout this section, we compare performance in terms of iterations of Alg. 1 for NGD and iterations of stochastic gradient descent, each of which achieves a single update of the parameters $(\mathbf{W}, \xi)$ in the respective algorithm based on a batch $\mathcal{B}$; this is the number of epochs multiplied by the number of batches.

Peaks problem

The peaks benchmark is a synthetic dataset for understanding the qualitative performance of classification algorithms (Haber and Ruthotto 2017). Here, a scattered point cloud in the two- dimensional unit square $[0, 1]^2$ is partitioned into disjoint sets. The classification problem is to determine which of those sets a given 2D point belongs to. The two-dimensional nature allows visualization of how NGD and GD classify data. In particular, plots of both how the nonlinear basis encoded by the hidden layer maps onto classification space and how the linear layer combines the basis functions to assign a probability map over the input space are readily obtained. We train a depth 4 dense network of the form (1) with $N_{\text{in}} = 2$, three hidden layers of width 12 contracting to a final hidden layer of width $N_{\text{basis}} = 6$, with $\tanh$ activation and batch normalization, and $N_{\text{classes}} = 5$ classes. As specified by Haber and Ruthotto (2017), 5000 training points are sampled from $[0, 1]^2$. The upper-left most image in Figure 2 shows the sampled data points with their observed classes. For the peaks benchmark we use a single batch containing all training points, i.e. $\mathcal{B} = \mathcal{D}$. The NGD algorithm uses 5 Newton iterations per training step with 3 CG iterations approximating the linear solve. The learning rate for Adam for both NGD and GD is 10^{-4}.

Figure 1 demonstrates that for an identical architecture, NGD provides a rapid increase in both training and validation accuracy over GD after a few iterations. For a large number of iterations both approaches achieve comparable training accuracy, although NGD generalizes better to the validation set. The improvement in validation accuracy is borne out in Figure 2, which compares representative instances of training using GD and NGD. While a single instance is shown, the character of these results is consistent with other neural networks trained for the Peaks problem in the same way. The top row illustrates the predicted classes argmax $[\mathcal{F}_{\text{SM}}(\mathbf{x})] \in \{0, 1, 2, 3, 4\}$ for $\mathbf{x} \in [0, 1]^2$ and the

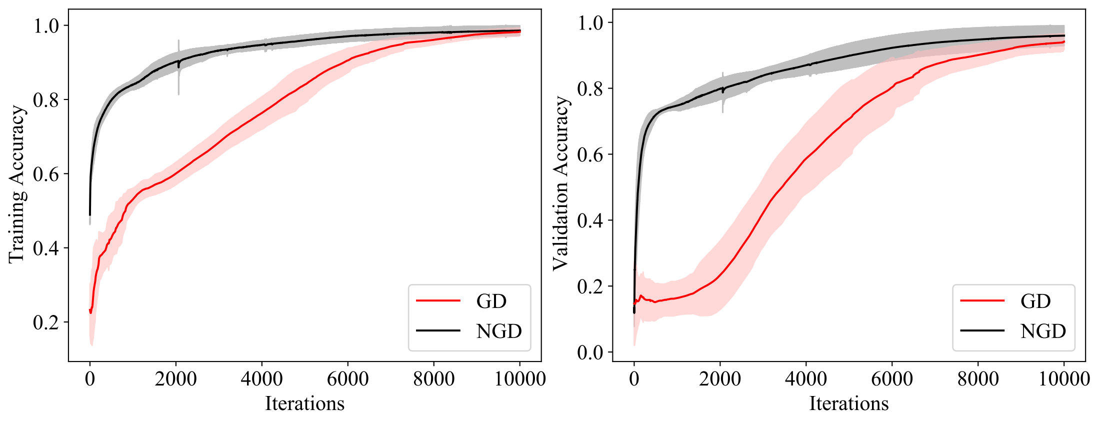

Figure 1: The training (*left*) and validation (*right*) accuracy for the peaks problem for both gradient descent (GD) and the Newton/gradient descent (NGD) algorithm. The solid lines represent the mean of 16 independent runs, and the shaded areas represent the mean $\pm$ one standard deviation.

training data, demonstrating that the NGD-trained network predicts the class $i = 2$ of lowest training point density more accurately than the GD-trained network. The remaining sets of images visualize both the classification probability map $[\mathcal{F}_{\text{SM}}(\mathbf{x})]_i$ for $i \in \{0, 1, 2, 3, 4\}$ (middle row) and the six basis functions $\Phi_\alpha(\mathbf{x}, \xi)$ (bottom row) learned by each optimizer. The difference in the learned bases is striking. GD learns a basis that is nearly discontinuous, in which the support of each basis function appears fit to the class boundaries. On the other hand, NGD learns a far smoother basis that can be combined to give sharper class boundary predictions. This manifests in the resulting probability map assigned to each class; linear combinations of the rougher GD basis results in imprinting and assignment of probability far from the relevant class. This serves as an explanation of the improved validation accuracy of NGD as compared to GD despite similar final training accuracy. The NGD algorithm separates refinement of the basis from the determination of the coefficients. This provides an effective regulation of the final basis, leading to improved generalization.

Image recognition benchmarks

We consider in this section a collection of image classification benchmarks: MNIST (Deng 2012; Grother 1995), fashion MNIST (Xiao, Rasul, and Vollgraf 2017), and CIFAR-10 (Krizhevsky, Hinton et al. 2009). We focus primarily on CIFAR-10 due to its increased difficulty; it is well-known that one may obtain near-perfect accuracy in the MNIST benchmark without sophisticated choice of architecture. For all problems, we consider a simple dense network architecture to highlight the role of the optimizer, and for CIFAR-10 we also utilize convolutional architectures (ConvNet). This highlights that our approach applies to general hidden layer architectures. Our objective is to demonstrate improvements in accuracy due to optimization with all other aspects held equal. For CIFAR-10, for example, the state-of-the-art requires application of techniques such as data-augmentation and complicated architectures to realize good accuracy; for simplicity we do not consider such complications to allow a simple comparison. The code for this study is provided at github.com/rgp62/.

For all results reported in this section, we first optimize the hyperparameters listed in Table 1 by maximizing the validation accuracy over the training run. We perform this search using the Gaussian process optimization tool in the scikit-optimize package with default options (Head et al. 2018). This process is performed for both GD and NGD to allow a fair comparison. The ranges for the search are shown in Table 1 with the optimal hyperparameters for each dataset examined in this study. For all problems we partition data into training, validation and test sets to ensure hyperparameter optimization is not overfitting. For MNIST and fashion MNIST we consider a $50K/10K/10K$ partition, while for CIFAR-10 we consider a $40K/10K/10K$ partition. All training is performed with a batch size of 1000 over 100 epochs. For all results the test accuracy falls within the first standard deviation error bars included in Figure 3.

Figure 3 shows the training and validation accuracies using the optimal hyperparameters for a dense architecture with two hidden layers of width 128 and 10 and ReLU activation functions. We find for all datasets, NGD more quickly reaches a maximum validation accuracy compared to GD, while both optimizers achieve a similar final validation accuracy. For the more difficult CIFAR-10 benchmark, NGD attains the maximum validation accuracy of GD in roughly one-quarter of the number of iterations. In Figure 3, we also use the CIFAR-10 dataset to compare the dense architecture to the following ConvNet architecture,

$$\underset{\text{8 channels, 3x3 kernel}}{\text{Convolution}} \rightarrow \underset{\text{2x2 window}}{\text{Max Pooling}} \rightarrow \underset{\text{16 channels, 3x3 kernel}}{\text{Convolution}}$$
$$\rightarrow \underset{\text{2x2 window}}{\text{Max Pooling}} \rightarrow \underset{\text{16 channels, 3x3 kernel}}{\text{Convolution}} \rightarrow \underset{\text{width 64}}{\text{Dense}} \rightarrow \underset{\text{width 10}}{\text{Dense}}$$

where the convolution and dense layers use the ReLU activation function. Again, NGD attains the maximum validation accuracy of GD in one-quarter the number of iterations, and also leads to an improvement of 1.76% in final test accuracy. This illustrates that NGD accelerates training and can improve accuracy for a variety of architectures.

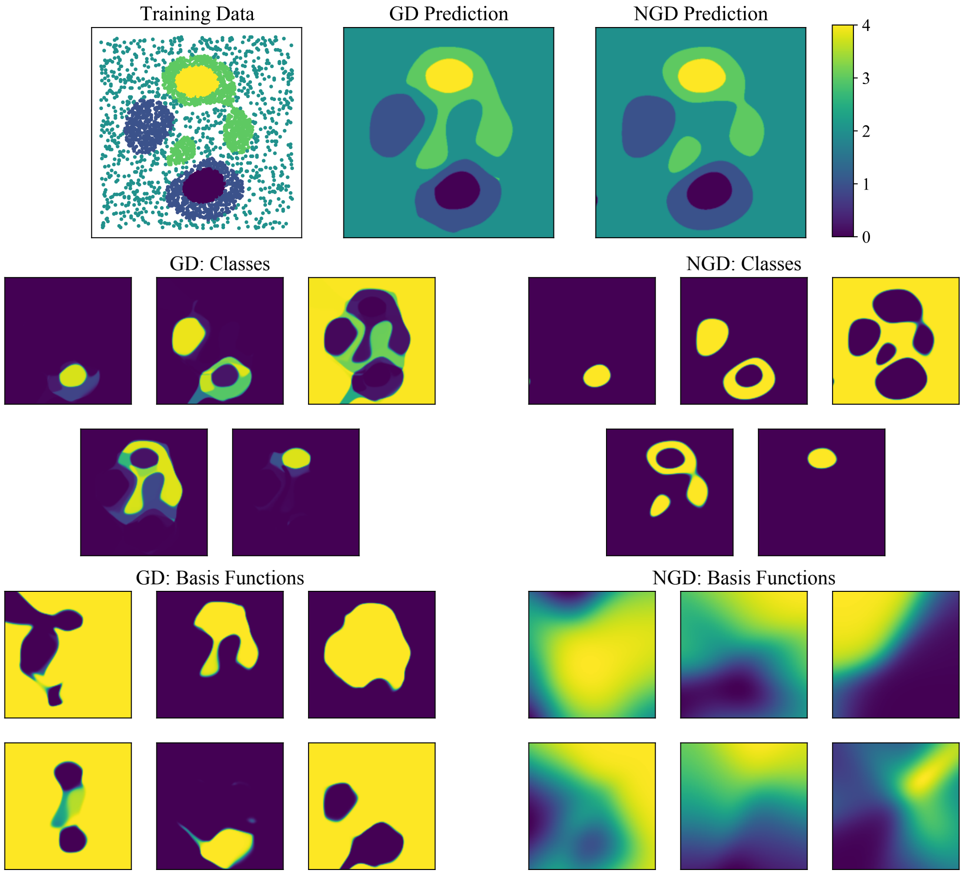

Figure 2: Results for peaks benchmarks, with comparison between NGD and GD on an identical architecture. In this example, GD obtained a training accuracy of 99.3% and validation accuracy of 96.2%, while NGD obtained a training accuracy of 99.6% and validation accuracy of 98.0%. **Top:** Training data (*left*), classification by GD (*center*), and classification by NGD (*right*). GD misclassifies large portions of the input space. **Middle:** The linear and softmax layers combine basis functions to assign classification probabilities to each class. The sharp basis learned in GD leads to artifacts and attribution of probability far from the sets (*left*) while diffuse basis in NGD provides a sharp characterization of class boundaries (*right*). **Bottom:** Adaptive basis encoded by hidden layer, as learnt by GD (*left*) and NGD (*right*). For GD the basis is sharp, and individual basis functions conform to classification boundaries, while NGD learns a more regular basis.

Hyperparameter	range	MNIST	Fashion MNIST	CIFAR-10	CIFAR-10 ConvNet
Learning rate	$[10^{-8}, 10^{-2}]$	$10^{-2.81}$ $10^{-2.26}$	$10^{-3.33}$ $10^{-2.30}$	$10^{-3.57}$ $10^{-2.50}$	$10^{-2.66}$ $10^{-2.30}$
Adam decay parameter 1	$[0.5, 1]$	0.537 0.630	0.756 0.657	0.629 0.891	0.755 0.657
Adam decay parameter 2	$[0.5, 1]$	0.830 0.616	0.808 0.976	0.782 0.808	0.858 0.976
CG iterations	$[1, 10]$	3	1	2	2
Newton iterations	$[1, 10]$	6	5	4	7

Table 1: Hyperparameters varied in study (*first column*), ranges considered (*second column*), and optimal values found for MNIST (*third column*), Fashion MNIST (*fourth column*), CIFAR-10 (*fifth column*), and CIFAR-10 with the ConvNet architecture (*last column*). For the learning rate and the Adam decay parameters, the optimal values for NGD followed by GD are shown. The optimal CG and Newton iterations are only applicable to NGD.

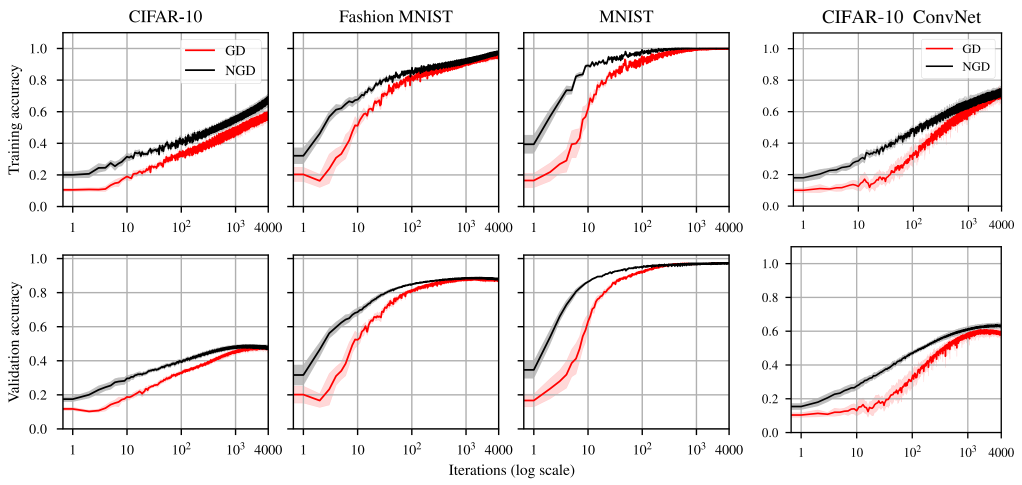

Figure 3: Training accuracy (*top row*) and validation accuracy (*bottom row*) for CIFAR-10, Fashion MNIST, and MNIST datasets using the dense architecture, and CIFAR-10 using the ConvNet architecture. Mean and standard deviation over 10 training runs are shown.

Conclusion

The NGD method, motivated by the adaptive basis interpretation of deep neural networks, is a block coordinate descent method for classification problems. This method separates the weights of the linear layer from the weights of the preceding nonlinear layers. NGD uses this decomposition to exploit the convexity of the cross-entropy loss with respect to the linear layer variables. It utilizes a Newton method to approximate the global minimum for a given batch of data before performing a step of gradient descent for the remaining variables. As such, it is a hybrid first/second order optimizer which extracts significant performance from a second-order substep that only scales with the number of weights in the linear layer, making it an appealing and feasible application of second-order methods for training very deep neural networks. Applying this optimizer to dense and convolutional networks, we have demonstrated acceleration with respect to the number of epochs in the validation loss for the peaks, MNIST, Fashion MNIST, and CIFAR-10 benchmarks, with improvements in accuracy for peaks benchmark and CIFAR-10 benchmark using a convolutional network.

Examining the basis functions encoded in the hidden layer of the network in the peaks benchmarks revealed significant qualitative difference between NGD and stochastic gradient descent in the exploration of parameter space corresponding to the hidden layer variables. This, and the role of the tolerance in the Newton step as an implicit regularizer, merit further study.

The difference in the regularity of the learned basis and probability classes suggests that one may obtain a qualitatively different model by varying only the optimization scheme used. We hypothesize that this more regular basis may have desirable robustness properties which may effect resulting model sensitivity. This could have applications toward training networks to be more robust against adversarial attacks.

Acknowledgements

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government. SAND Number: SAND2021-3013 C.

The work of R. Patel, N. Trask, and M. Gulian are supported by the U.S. Department of Energy, Office of Advanced Scientific Computing Research under the Collaboratory on Mathematics and Physics-Informed Learning Machines for Multiscale and Multiphysics Problems (PhILMs) project. E. C. Cyr is supported by the Department of Energy early career program. M. Gulian is supported by the John von Neumann fellowship at Sandia National Laboratories.

References

Armijo, L. 1966. Minimization of functions having Lipschitz continuous first partial derivatives. *Pacific Journal of mathematics* 16(1): 1–3.

Barratt, S. T.; and Boyd, S. P. 2020. Least squares auto-tuning. *Engineering Optimization* 1–22.

Blondel, M.; Seki, K.; and Uehara, K. 2013. Block coordinate descent algorithms for large-scale sparse multiclass classification. *Machine learning* 93(1): 31–52.

Botev, A.; Ritter, H.; and Barber, D. 2017. Practical Gauss-Newton optimisation for deep learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 557–565. JMLR. org.

Bottou, L. 2010. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, 177–186. Springer.

Bottou, L.; Curtis, F. E.; and Nocedal, J. 2018. Optimization methods for large-scale machine learning. *Siam Review* 60(2): 223–311.

Boyd, S.; Boyd, S. P.; and Vandenberghe, L. 2004. *Convex optimization*. Cambridge university press.

Bubeck, S. 2014. Convex optimization: Algorithms and complexity. *arXiv preprint arXiv:1405.4980* .

Clarkson, K. L.; Hazan, E.; and Woodruff, D. P. 2012. Sublinear optimization for machine learning. *Journal of the ACM (JACM)* 59(5): 1–49.

Cyr, E. C.; Gulian, M. A.; Patel, R. G.; Perego, M.; and Trask, N. A. 2019. Robust training and initialization of deep neural networks: An adaptive basis viewpoint. *arXiv preprint arXiv:1912.04862* .

Daubechies, I.; DeVore, R.; Foucart, S.; Hanin, B.; and Petrova, G. 2019. Nonlinear approximation and (deep) ReLU networks. *arXiv preprint arXiv:1905.02199* .

Deng, L. 2012. The MNIST database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine* 29(6): 141–142.

Dennis Jr, J. E.; and Schnabel, R. B. 1996. *Numerical methods for unconstrained optimization and nonlinear equations*, volume 16. Siam.

Folland, G. B. 1999. *Real analysis: modern techniques and their applications*, volume 40. John Wiley & Sons.

Fu, W. J. 1998. Penalized regressions: the bridge versus the lasso. *Journal of computational and graphical statistics* 7(3): 397–416.

Grother, P. J. 1995. NIST special database 19 handprinted forms and characters database. *National Institute of Standards and Technology* .

Haber, E.; and Ruthotto, L. 2017. Stable architectures for deep neural networks. *Inverse Problems* 34(1): 014004.

Head, T.; MechCoder; Louppe, G.; Shcherbatyi, I.; fcharras; Vinícius, Z.; cmmalone; Schröder, C.; nel215; Campos, N.; and et al. 2018. scikit-optimize/scikit-optimize: v0.5.1 - re-release doi:10.5281/zenodo.1170575.

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

Krizhevsky, A.; Hinton, G.; et al. 2009. Learning multiple layers of features from tiny images .

Martens, J. 2010. Deep learning via Hessian-free optimization. In *ICML*, volume 27, 735–742.

McLoone, S.; Brown, M. D.; Irwin, G.; and Lightbody, A. 1998. A hybrid linear/nonlinear training algorithm for feedforward neural networks. *IEEE Transactions on Neural Networks* 9(4): 669–684.

Nesterov, Y. 2012. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization* 22(2): 341–362.

Opschoor, J. A.; Petersen, P.; and Schwab, C. 2019. Deep ReLU networks and high-order finite element methods. *SAM, ETH Zürich* .

Osawa, K.; Tsuji, Y.; Ueno, Y.; Naruse, A.; Foo, C.-S.; and Yokota, R. 2020. Scalable and Practical Natural Gradient for Large-Scale Deep Learning. *arXiv preprint arXiv:2002.06015* .

Osawa, K.; Tsuji, Y.; Ueno, Y.; Naruse, A.; Yokota, R.; and Matsuoka, S. 2019. Large-scale distributed second-order optimization using Kronecker-factored approximate curvature for deep convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 12359–12367.

Pereyra, V.; Scherer, G.; and Wong, F. 2006. Variable projections neural network training. *Mathematics and Computers in Simulation* 73(1-4): 231–243.

Shevade, S. K.; and Keerthi, S. S. 2003. A simple and efficient algorithm for gene selection using sparse logistic regression. *Bioinformatics* 19(17): 2246–2253.

Wright, S. J. 2015. Coordinate descent algorithms. *Mathematical Programming* 151(1): 3–34.

Xiao, H.; Rasul, K.; and Vollgraf, R. 2017. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747* .

Xu, P.; Roosta, F.; and Mahoney, M. W. 2019. Newton-type methods for non-convex optimization under inexact hessian information. *Mathematical Programming* 1–36.

Yarotsky, D. 2017. Error bounds for approximations with deep ReLU networks. *Neural Networks* 94: 103–114.