An Investigation of ADAM: A Stochastic Optimization Method

Michael Pérez [1]

Abstract

This paper reviews and investigates *Adam* (Kingma & Ba, 2014), an algorithm for first-order optimization of stochastic objective functions. Specifically, the *Adam* and *AdaMax* optimization algorithms, three convex and non-convex classification models, and the dropout regularizer are implemented for classification of the MNIST dataset. The performance of these methods is analyzed through extensive comparisons. Using *Adam* to optimize a convolutional neural network yielded the highest classification accuracy on the MNIST test set, 99.21%.

1. Introduction

Stochastic gradient optimization is important in many science and engineering fields. In these fields, problems are commonly framed as minimization or maximization of an objective function with respect to its scalar parameters. If the objective function is differentiable with respect to its parameters, traditional gradient descent is an efficient optimization method. The objective function is often stochastic. Stochastic objective functions can take the form of a sum of subfunctions evaluated on different subsamples of data (minibatches). With a stochastic objective, the efficiency of optimization can be improved by taking gradient steps with respect to individual subfunctions, or stochastic gradient descent (SGD). SGD has been instrumental to recent advances in machine learning (Krizhevsky et al., 2012). Other sources can contribute noise to the objective, such as dropout (Hinton et al., 2012). The optimization of stochastic objective functions with high-dimensional parameter spaces is challenging.

Adam is an efficient first-order optimization algorithm. It computes individual adaptive learning rates from estimates of the first and second moments of the gradients, avoiding expensive higher order computations. The name Adam is derived from adaptive moment estimation. Adam is interesting because it is commonly used today as an alternative to SGD for training deep neural networks. It is useful because it handles sparse gradients, is memory efficient, and is well-suited for non-convex optimization problems. Adam builds upon RMSProp (Tieleman et al., 2012; Graves, 2013) and AdaGrad (Duchi et al., 2011).

[*]Equal contribution [1]Computer and Information Science and Engineering Department, University of Florida, Gainesville, USA. Correspondence to: Michael Pérez <michaelperez012@ufl.edu>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

2. Problem Statement

Adam and AdaMax will be used to optimize three models: multi-class logistic regression (LR), a multi-layer neural network (NN), and a convolutional neural network (CNN)

2.1. Quadratically Regularized Multi-Class Logistic Regression

Algorithms for multi-class logistic regression solve the following optimization problem:

$$\underset{\boldsymbol{\Theta}}{\text{minimize}} \frac{1}{n}\sum_{i=1}^{n}(\log(\sum_{c=1}^{k}\exp \boldsymbol{x}_i^T\boldsymbol{\theta}_c) - x_i^T\boldsymbol{\theta}_{y_i}) + \lambda\|\boldsymbol{\Theta}\|^2$$

2.2. Multi-layer Neural Network

The multi-layer neural network architecture that is optimized in experiments has two fully-connected hidden layers with 1000 hidden units each, and uses the ReLU activation function. Neural networks generally solve the following optimization problem:

$$\underset{\boldsymbol{W}_1,\ldots,\boldsymbol{W}_Q,\boldsymbol{\Theta}}{\text{minimize}} \sum_{i=1}^{n}\ell\left(\boldsymbol{y}_i, \boldsymbol{\Theta}^\top a\left(\boldsymbol{W}_Q a\left(\cdots a\left(\boldsymbol{W}_1\boldsymbol{x}_i\right)\right)\right)\right) +$$

$$\lambda\sum_{q=1}^{Q}\|\boldsymbol{W}_q\|^2 + \lambda\|\boldsymbol{\Theta}\|^2$$

The ReLU activation function has the form $a(u) = \max(0, a)$. ℓ is the loss function. In this work cross-entropy loss is used: $\ell\left(\boldsymbol{y}_i, \boldsymbol{\Theta}^\top a\left(\boldsymbol{W}_Q a\left(\cdots a\left(\boldsymbol{W}_1\boldsymbol{x}_i\right)\right)\right)\right) = -\log\frac{\exp a\left(\boldsymbol{W}_Q a\left(\cdots a\left(\boldsymbol{W}_1\boldsymbol{x}_{y_i}\right)\right)\right)}{\sum_{c=1}^{k}\exp a\left(\boldsymbol{W}_Q a\left(\cdots a\left(\boldsymbol{W}_1\boldsymbol{x}_c\right)\right)\right)}$.

2.3. Convolutional Neural Network

In a convolutional neural network, the formulation is the same as the multi-layer case, except the weights in at least

Algorithm 1 Adam. All vector operations are element-wise.

Input: α: Stepsize
Input: ϵ: Machine precision threshold
Input: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Input: $f(\theta)$: Stochastic objective function with parameters θ
Input: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1st moment vector)
$v_0 \leftarrow 0$ (Initialize 2nd moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
  $t \leftarrow t + 1$
  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
  $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$ (Update biased first moment estimate)
  $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$ (Update biased second raw moment estimate)
  $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
  $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
  $\theta_t \leftarrow \theta_{t-1} - \frac{\alpha \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$ (Update parameters)
end while
return θ_t (Resulting parameters)

Algorithm 2 AdaMax. All vector operations are element-wise.

Input: α: Stepsize
Input: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Input: $f(\theta)$: Stochastic objective function with parameters θ
Input: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1st moment vector)
$u_0 \leftarrow 0$ (Initialize the exponentially weighted infinity norm)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
  $t \leftarrow t + 1$
  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
  $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$ (Update biased first moment estimate)
  $u_t \leftarrow \max(\beta_2 u_{t-1}, |g_t|)$ (Update the exponentially weighted infinity norm)
  $\theta_t \leftarrow \theta_{t-1} - \frac{\alpha}{(1-\beta_1^t)} \frac{m_t}{u_t}$ (Update parameters)
end while
return θ_t (Resulting parameters)

one convolutional layer make up kernels that are convolved over the input.

$$\underset{\boldsymbol{W}_1,\ldots,\boldsymbol{W}_Q,\boldsymbol{\Theta}}{\text{minimize}} \sum_{i=1}^{n} \ell\left(\boldsymbol{y}_i, \boldsymbol{\Theta}^\top a\left(\boldsymbol{W}_Q * a\left(\cdots a\left(\boldsymbol{W}_1 * \boldsymbol{x}_i\right)\right)\right)\right)$$

$$+\lambda \textstyle\sum_{q=1}^{Q} \|\boldsymbol{W}_q\|^2 + \lambda \|\boldsymbol{\Theta}\|^2$$

3. Algorithms

3.1. Adam

Adam proceeds as shown in Algorithm 1. Exponentially decaying averages of the gradient (m_t) and squared gradient (v_t) are updated with β_1 and β_2 as hyper-parameters that control the decay rate. These averages are estimates of the first (mean) and second moments (uncentered variance) of the gradient. The moving averages are initialized to **0**'s, which can lead to moment averages that are biased towards zero. This initialization bias is countered by calculating the bias-corrected estimates $\hat{m}_t$ and $\hat{v}_t$ The efficiency of Algorithm 1 can be improved by replacing the last three lines with: $\alpha_t = \frac{\alpha\sqrt{1-\beta_2^t}}{1-\beta_1^t}$ and $\theta_t \leftarrow \theta_{t-1} - \frac{\alpha_t m_t}{\sqrt{v_t}+\hat{\epsilon}}$.

Adam is closely related to RMSProp (Tieleman et al., 2012) and RMSProp with momentum (Graves, 2013), optimization algorithms designed for non-stationary objectives. In RMSProp with momentum the parameters are updated using momentum on the rescaled gradient. In Adam, the updates are directly computed using a running average of first and second moments of the gradient. RMSProp also does not have a bias correction term, which can lead to divergence in some cases.

Adam is also related to AdaGrad (Duchi et al., 2011), an optimization algorithm designed for dealing with sparse gradients. AdaGrad updates the parameters using $\theta_{t+1} = \theta_t - \frac{\alpha g_t}{\sum_{i=1}^{t} \sqrt{g_t^2}}$ (element-wise division). AdaGrad is a version of Adam with $\beta_1 = 0$, infinitesimally small β_2, and a different step size, $\alpha_t = \frac{\alpha}{\sqrt{t}}$. AdaGrad uses the Mahalanobis norm $\|\theta\|_{G^{(t)1/2}} = \theta^T G^{(t)1/2} \theta$ as the distance metric instead of $\|\theta - \theta^{(t)}\|$. Computing $(G^{(t)})^{-1/2}$ is expensive so a diagonal approximation is used.

3.1.1. Why does Adam work?

There are mathematical problems in the convergence proof of Adam in the convex case presented in the original Adam paper. (Reddi et al., 2019) A recent work (Défossez et al., 2020) provides a correct convergence proof for the Adam and Adagrad optimization algorithms when applied to smooth (potentially non-convex) objective functions with bounded gradients. The authors show that in expectation, the squared norm of the objective gradient averaged over the trajectory can be upper bounded in terms of the constants,

optimizer parameters, dimensions d, and total iterations N. With the right hyperparameters the bound can be made arbitrarily small and the algorithm is shown to converge at the same rate as in the original paper, $O(d\ln(N)/\sqrt{N})$. Adam does not converge with the default hyperparameters (Kingma & Ba, 2014), yet still moves away from the initialization point faster than AdaGrad, explaining its practical success.

Adam works intuitively because it aims to combine the advantages of two other extensions of stochastic gradient descent, AdaGrad and RMSProp. AdaGrad incorporates a per-parameter learning rate that improves performance for problems with sparse gradients. RMSProp has adaptive learning rates and updates them based on estimates of the mean and variance of the gradients, which is helpful for online and noisy problems.

3.2. AdaMax

AdaMax is an extension to Adam based on the infinity norm. In Adam individual weights are updated by scaling their gradients inversely proportional to a scaled L^2 norm of their current and past gradients. Norms for p values greater than two are generally numerically unstable, except for the infinity norm. If the update rule based on the L^2 norm is generalized to a rule based on the L^∞ norm, a simple algorithm is produced, AdaMax. AdaMax does not need bias-correction. See Algorithm 2 for pseudocode.

3.2.1. Why does AdaMax work?

AdaMax has shown good performance in practice but does not yet have theoretical convergence guarantees (Chakrabarti & Chopra, 2021). It works intuitively for the same reasons that AdaMax works, except using an update rule based on the L^∞ norm.

4. Experiments

The algorithms explained were implemented on different machine learning models to evaluate their efficacy in Python. In all implementations algorithms were trained for 20 epochs, with a decaying step size $\alpha_t = \frac{\alpha}{\sqrt{t}}$ and a mini-batch size of 128. The exponential decay rates and machine precision threshold hyperparameters were set as follows: $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$. Similar values were used in the Adam paper (Kingma & Ba, 2014). The neural networks were implemented with and without dropout, and were optimized using both Adam and AdaMax. The regularization hyperparameter λ is set to 0 in all experiments because it did not make a noticeable difference.

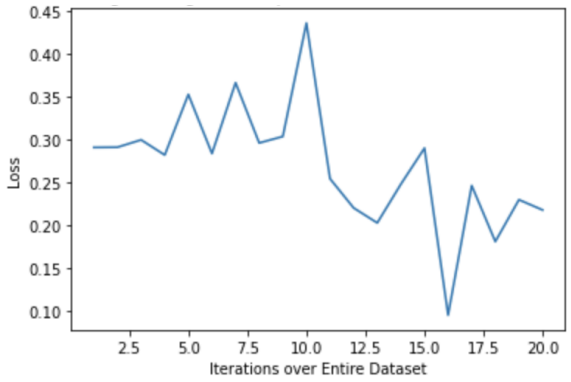

Figure 1. Multi-class Logistic Regression optimized with Adam

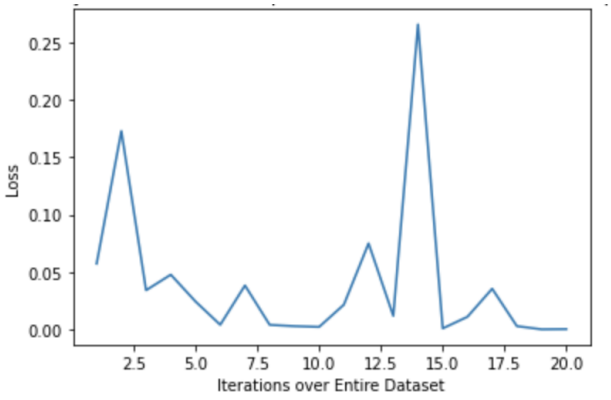

Figure 2. Multi-layer Neural Network optimized with Adam, without dropout

4.1. Adam

4.1.1. Quadratically Regularized Multi-Class Logistic Regression

Algorithm 1 was implemented on L^2-regularized multiclass logistic regression on the MNIST dataset (Deng, 2012). The implementation solves the logistic regression formulation in Section 2.1. An accuracy of 92.69% was achieved on the MNIST test set. See Figure 1.

4.1.2. Multi-layer Neural Network

Algorithm 1 was implemented on a multi-layer neural network on the MNIST dataset. The NN uses the ReLU activation function. The implementation solves the multi-layer neural network optimization formulation in Section 2.2. An accuracy of 98.17% was achieved on the MNIST test set without dropout. With dropout, 98.43% was achieved. Figure 2 shows the loss over iterations for the implementation without dropout.

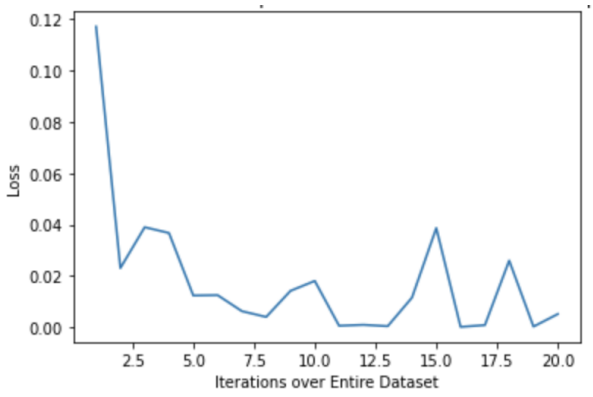

Figure 3. Convolutional Neural Net optimized with Adam, without dropout

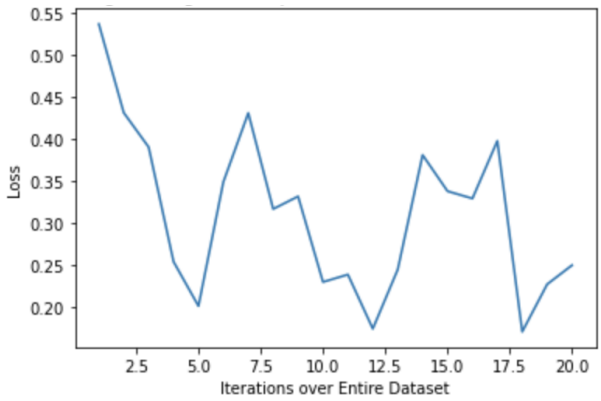

Figure 4. Multi-class Logistic Regression optimized with AdaMax

4.1.3. Convolutional Neural Network

Algorithm 1 was implemented on a convolutional neural network on the MNIST dataset. The CNN has two alternating stages of a 5×5 convolutional layer, a 3×3 max pooling layer with a stride of 2, and a ReLU layer. Following those two stages there are two fully-connected layers. The implementation solves the convolutional neural network optimization formulation in Section 2.3. An accuracy of 99.21% was achieved on the MNIST test set without dropout. With dropout, 99.21% was achieved. Figure 3 shows the loss over iterations for the implementation without dropout.

4.2. AdaMax

4.2.1. Quadratically Regularized Multi-Class Logistic Regression

Algorithm 2 was implemented on L^2-regularized multiclass logistic regression on the MNIST dataset. The implementation solves the logistic regression formulation in Section 2.1. An accuracy of 92.38% was achieved on the MNIST test set. See Figure 4.

4.2.2. Multi-layer Neural Network

Algorithm 2 was implemented on a NN for classification of the MNIST dataset. The neural network architecture has two fully-connected hidden layers with 1000 hidden units each. The ReLU activation function was used. The implementation solves the multi-layer neural network optimization formulation in Section 2.2. An accuracy of 98.33% was achieved on the MNIST test set without dropout. With dropout, 98.14% was achieved. Figure 5 shows the loss over iterations for the implementation without dropout.

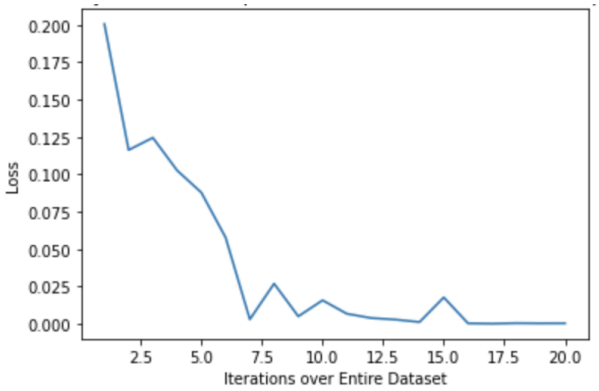

Figure 5. Multi-layer Neural Net optimized with AdaMax, without dropout

4.2.3. Convolutional Neural Network

Algorithm 2 was implemented on a CNN for classification of the MNIST dataset. The CNN has two alternating stages of a 5×5 convolutional layer, a 3×3 max pooling layer with a stride of 2, and a ReLU layer. Following the two stages there are two fully-connected layers. The implementation solves the convolutional neural network optimization formulation in Section 2.3. An accuracy of 99.09% was achieved on the MNIST test set without dropout. With dropout, 99.09% was achieved. Figure 6 shows the loss over iterations for the implementation without dropout.

4.3. Comparisons

4.3.1. Dropout vs. no Dropout in NN and CNN with Adam and AdaMax

I investigated the use of dropout because dropout adds stochastic regularization in order to reduce overfitting. Figures 7 and 8 show a comparison of the classification perfor-

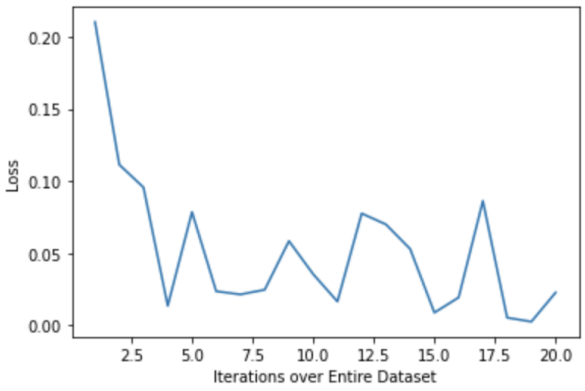

Figure 6. Convolutional Neural Net optimized with AdaMax, without dropout

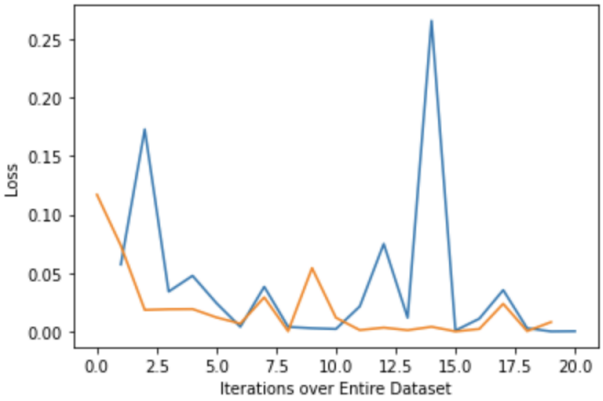

Figure 7. Dropout (orange) vs. no Dropout (blue) in Multi-layer Neural Network with Adam

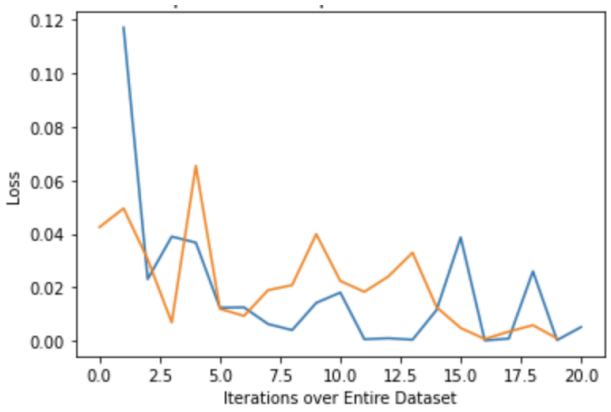

Figure 8. Dropout (orange) vs. no Dropout (blue) in Convolutional Neural Network with Adam

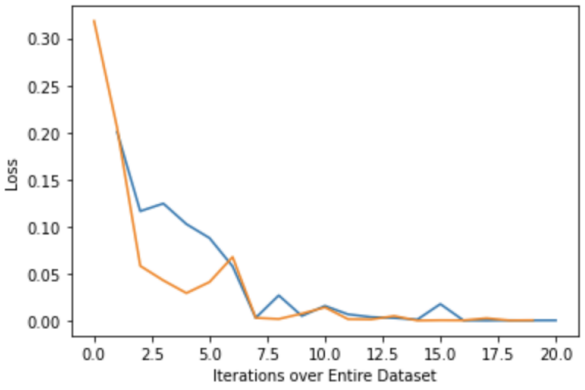

Figure 9. Dropout (orange) vs. no Dropout (blue) in Multi-layer Neural Network with AdaMax

mance of the neural network architectures with and without dropout, optimized with Adam. Figures 9 and 10 show a comparison of the classification performance of the neural network architectures with and without dropout, optimized with AdaMax.

4.3.2. Adam vs. AdaMax in Logistic Regression, NN, and CNN

I also compared the classification performance of Adam and AdaMax when used to optimize the same models. Figures 11, 12, and 13 show a comparison of Adam and AdaMax when used to optimize logistic regression, a NN, and a CNN, respectively.

4.3.3. Logistic Regression vs. NN vs. CNN in Adam and AdaMax

I compared Adam and AdaMax's performance when used to optimize the three models described for classification of the MNIST dataset. Figure 14 shows a comparison of Adam's performance across the three models and Figure 15 shows a comparison of AdaMax's performance across the three models.

5. Conclusion

Table 1 summarizes the classifications accuracies of the different objective functions and optimization algorithms on the MNIST test set. Based on those results, using *Adam* to optimize a CNN performed best (with and without dropout performed equally well). There still remain conclusions to be derived from the results about dropout, the optimization algorithm, and the three objective functions. Based on Figures 7 - 10 dropout seems to generally stabilize training and provide regularization when used in neural networks. Based on Figures 11 - 13 *Adam* generally performs better than *AdaMax*. Based on Figures 14 - 14 neural networks evidently capture the nonlinearities in the MNIST

	LR	NN with dropout	NN without dropout	CNN with dropout	CNN without dropout
Adam	92.69%	98.43%	98.17%	99.21%	99.21%
AdaMax	92.38%	98.14%	98.33%	99.09%	99.09%

Table 1. Summary of Classification Accuracies on MNIST Test Set

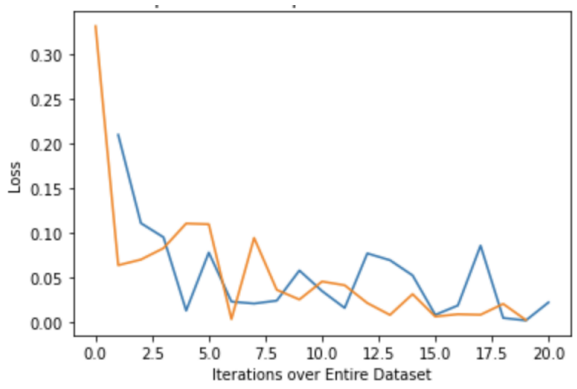

Figure 10. Dropout (orange) vs. no Dropout (blue) in Convolutional Neural Network with AdaMax

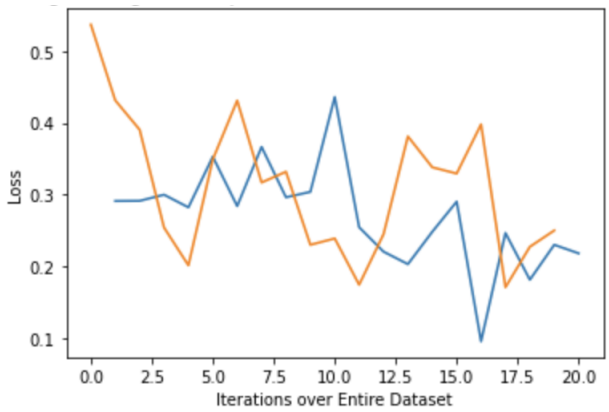

Figure 11. Multi-class Logistic Regression optimized with Adam (blue) vs. AdaMax (orange)

dataset much better than logistic regression. Interestingly, the CNN performed better than the NN when optimized using *Adam*, but the converse relationship was observed when optimized using *AdaMax*. One traditionally expects the convolutional filters in a CNN to process images better than a fully-connected NN. This discrepancy can be attributed to how the second moment estimate in *Adam* is a worse approximation of the geometry of the cost function in CNNs compared to fully-connected NNs (Kingma & Ba, 2014).

At the time the Adam paper was published, the convex convergence proof was not completely correct and a nonconvex convergence proof did not exist. Since then, other publications have presented a correct, generic convergence proof with optimal hyperparameter values. Nadam (Nesterov-accelerated Adaptive Moment Estimation) (Dozat, 2016) aims to combine Adam and Nesterov's accelerated gradient by modifying Adam's momentum component. Convergence proofs of AdaMax and Nadam remain an open questions.

Besides the theoretical proofs, there do not remain many open questions in the field of gradient-based optimization, as far as I know. The field is sufficiently developed such that a machine learning engineer can design a feature representation, choose an objective function that models the data distribution, and then choose an optimization algorithm depending on certain criteria (convex vs. nonconvex, sparse vs. dense gradient, etc.). Lastly, the hyperparameter values should be tuned by training on a training set and benchmarking performance on a validation set. I certainly expect that there are more open problems for specific applications. Adam is already used effectively in practice to solve large-scale, high dimensional machine learning problems.

References

Chakrabarti, K. and Chopra, N. Generalized adagrad (g-adagrad) and adam: A state-space perspective. In *2021 60th IEEE Conference on Decision and Control (CDC)*, pp. 1496–1501, 2021. doi: 10.1109/CDC45484.2021.9682994.

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

Dozat, T. Incorporating nesterov momentum into adam. 2016.

Duchi, J., Hazan, E., and Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12 (61):2121–2159, 2011. URL http://jmlr.org/papers/v12/duchi11a.html.

Défossez, A., Bottou, L., Bach, F., and Usunier, N. A simple convergence proof of adam and adagrad, 2020. URL https://arxiv.org/abs/2003.02395.

Graves, A. Generating sequences with recurrent neural

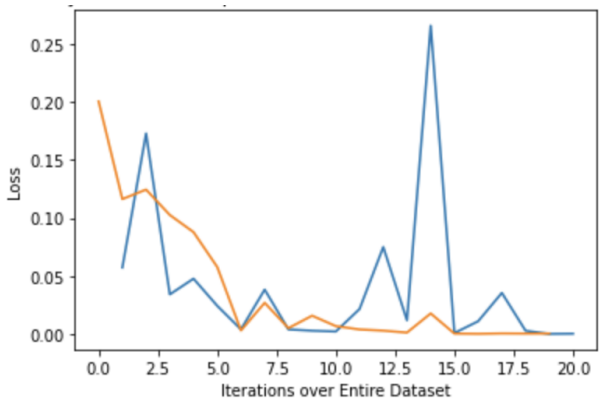

Figure 12. Multi-layer Neural Net optimized with Adam (blue) vs. AdaMax (orange)

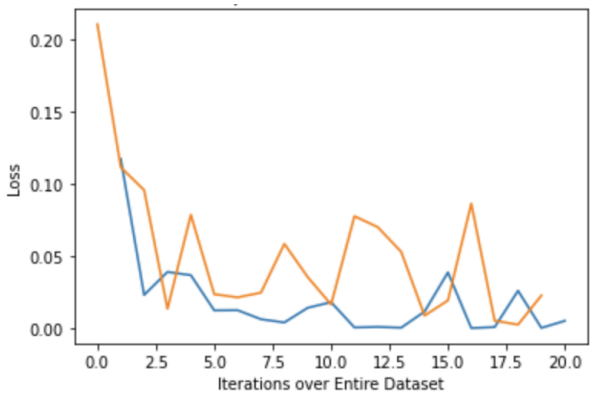

Figure 13. Convolutional Neural Net optimized with Adam (blue) vs. AdaMax (orange)

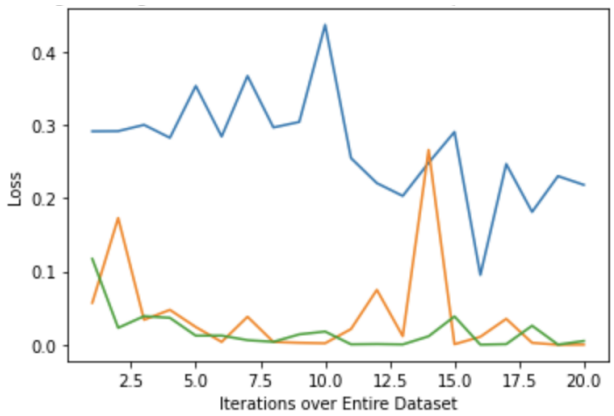

Figure 14. Logistic Regression (blue) vs. NN (orange) vs. CNN (green) optimized with Adam

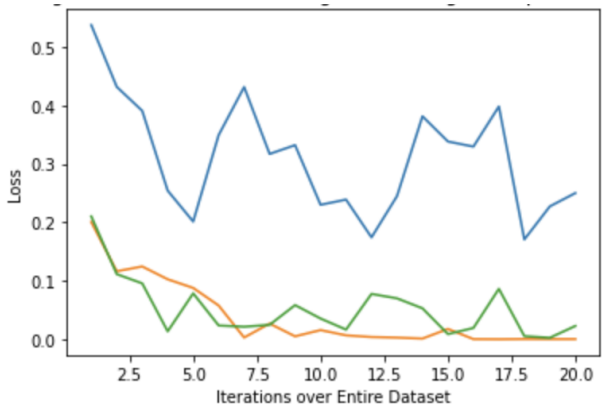

Figure 15. Logistic Regression (blue) vs. NN (orange) vs. CNN (green) optimized with AdaMax

networks. *CoRR*, abs/1308.0850, 2013. URL http://arxiv.org/abs/1308.0850.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. Improving neural networks by preventing co-adaptation of feature detectors, 2012. URL https://arxiv.org/abs/1207.0580.

Kingma, D. and Ba, J. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q. (eds.), *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

Reddi, S. J., Kale, S., and Kumar, S. On the convergence of adam and beyond, 2019. URL https://arxiv.org/abs/1904.09237.

Tieleman, T., Hinton, G., et al. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4 (2):26–31, 2012.