OPT2020: 12th Annual Workshop on Optimization for Machine Learning

Deep Residual Partitioning

Neal Lawton NLAWTON@USC.EDU
Greg Ver Steeg GREGV@ISI.EDU
Aram Galstyan GALSTYAN@ISI.EDU
Information Sciences Institute, Marina Del Rey, CA / University of Southern California, Los Angeles, CA

Abstract

Deep neural networks are trained by solving a hard optimization problem. This optimization problem can be effectively solved using general optimization methods like Adam, but we can exploit the special structure of neural networks to derive even better training algorithms. We introduce residual partitioning, a novel second-order optimization method for training neural nets. In each training iteration, residual partitioning uses Jensen's inequality to bound the objective; this bound has a diagonal Hessian and so is simple to optimize. In our experiments, we compare residual partitioning with several standard optimization algorithms on several machine learning tasks and conclude residual partitioning converges to a competitive or better solution.

1. Introduction

Training deep neural networks is a difficult optimization problem. This problem can be solved using gradient descent, which incrementally moves the model parameters in the direction of the gradient and uses a learning rate to determine how far. Since the curvature of the objective function varies greatly along different directions, using a constant learning rate causes the process to ping-pong along directions of high curvature, leading to slow convergence. Instead, fast optimization methods estimate the curvature of the objective around the current solution and adjust the learning rate accordingly, decreasing the learning rate along dimensions with high curvature and increasing the learning rate along dimensions with low curvature.

Adaptive learning rate methods such as Adam [10], AdaGrad [8], and AdaDelta [26] estimate the local curvature from an exponential moving average of past gradients using different heuristics. Since implementing these methods only requires efficient computation of the gradient, they can be used to solve a wide variety of optimization problems.

Second-order methods estimate the curvature using the Hessian matrix of second derivatives. Hessian-free learning uses the special structure of neural networks to efficiently compute Hessian-vector products [14, 17, 20, 22, 24]. Many second-order methods use a structured approximation of the Hessian matrix, e.g. low-rank [4, 13, 23], diagonal [2], or block-diagonal [6], to decrease wall time per training step or memory utilization. Since the Hessian typically exhibits negative curvature [18], many second-order methods make a positive-definite approximation to the Hessian, e.g., the Gauss-Newton approximation [3], the Fisher Information Matrix [1, 9, 15, 16, 19, 21], or by identifying and bounding the negative eigenvalues of the Hessian [5, 7]. In practice, modern second-order methods are still too slow, too memory-intensive, and too complicated to compete against adaptive gradient methods.

In this paper we introduce *residual partitioning*, a novel second-order optimization algorithm for training neural networks. In each training iteration, residual partitioning uses Jensen's inequality to construct an upper bound on the local second-order approximation of the loss function. This is a strong theoretical claim, since it guarantees that the decrease in loss achieved by minimizing the bound will also be achieved for the local second-order approximation. The bound we derive has a diagonal Hessian matrix and is therefore easy to optimize and memory efficient. This also makes it exceedingly simple to correct for negative curvature introduced by the non-linear activation functions without computing the eigendecomposition of any dense matrix. We can efficiently construct and minimize our bound with a single forward and backward pass through the network and in the same time complexity as an Adam or SGD step.

In our experiments, we compare residual partitioning with Adam and SGD by training an autoencoder and classifier network on MNIST, Fashion-MNIST [25], and CIFAR-10 [12]. We observe that residual partitioning converges to a competitive or better solution compared to Adam and SGD.

2. Notation and Setup

We use subscript ℓ or superscript (ℓ) to index layers, subscript s to index dataset samples, and subscript i or j to index neurons in a layer. The ℓ-th layer of a neural net with L layers has m_ℓ neurons with activations $\{y_{sj}^{(\ell)}\}_{j=1}^{m_\ell}$. Each affine layer has weights $\{w_{ij}\}$ and biases $\{b_i\}$, though we write each bias b_i as a weight w_{i0} with incoming activation $y_{s0} \equiv 1$. Each activation layer has an activation function $\sigma_\ell(\cdot) : \mathbb{R} \to \mathbb{R}$.

Denote $P \subseteq [L]$ as the set of indices corresponding to layers with trainable parameters. In each training step, for each $\ell \in P$, we update each weight $w_{ij}^{(\ell)}$ by adding on $\Delta w_{ij}^{(\ell)}$. The new weight is $\hat{w}_{ij}^{(\ell)} \equiv w_{ij}^{(\ell)} + \Delta w_{ij}^{(\ell)}$. Updating the weights causes the activation $y_{sj}^{(\ell)}$ to change; denote this change $\Delta y_{sj}^{(\ell)}$, so the activation after updating weights is $\hat{y}_{sj}^{(\ell)} \equiv y_{sj}^{(\ell)} + \Delta y_{sj}^{(\ell)}$. Residual partitioning is a method for solving optimization problems of a special form:

$$\min_{\{\Delta w_{ij}^{(\ell)}\}} \mathcal{L} \qquad\qquad \mathcal{L} \equiv \sum_{s=1}^{S} f(\hat{\boldsymbol{y}}_s^{(L)} | \boldsymbol{x}_s) \tag{1}$$

where $\{\boldsymbol{x}_s\}_{s=1}^S$ is a dataset of observations, $\hat{\boldsymbol{y}}_s^{(L)}$ is the output of a neural net consisting of some combination of activation layers and affine layers, and $f(\hat{\boldsymbol{y}}_s^{(L)}|\boldsymbol{x}_s) : \mathbb{R}^{m_L} \to \mathbb{R}$ is a convex loss function of $\hat{\boldsymbol{y}}_s^{(L)}$, such as the L^2 loss or cross-entropy error.

3. Residual Partitioning

To bound the second-order approximation to the objective function $\tilde{\mathcal{L}} \approx \mathcal{L}$, we will peel back the neural network layer by layer working backwards. Formally, we will prove the following with induction:

Theorem 1 *For each $\ell \in [L]$, there exists an upper bound $\tilde{\mathcal{L}}_\ell$ on $\tilde{\mathcal{L}}$ of the following form, where $\mathcal{Y}_{sj}^{(\ell)}$ and $\mathcal{W}_{ij}^{(\ell)}$ are univariate convex quadratic functions of $\Delta y_{sj}^{(\ell)}$ and $\Delta w_{ij}^{(\ell)}$, respectively:*

$$\tilde{\mathcal{L}} \leq \tilde{\mathcal{L}}_\ell \equiv \sum_{s=1}^{S} \sum_{j=1}^{m_\ell} \mathcal{Y}_{sj}^{(\ell)} + \sum_{\ell' \in P,\ \ell' \geq \ell} \sum_{i=1}^{m_{\ell'+1}} \sum_{j=1}^{m_{\ell'}} \mathcal{W}_{ij}^{(\ell')} \tag{2}$$

However, $\tilde{\mathcal{L}}_\ell$ is only an intermediary bound, with $\mathcal{W}_{ij}^{(\ell)}$ terms for parameters that have already been assigned part of the residual and $\mathcal{Y}_{sj}^{(\ell)}$ terms for the remaining pieces of the residual that will continue to backpropagate up to parameters in higher layers. The bound we are ultimately interested in is $\tilde{\mathcal{L}}_1$, where the quadratic function $\mathcal{Y}_{sj}^{(1)}$ is a constant (since $\Delta y_{sj}^{(1)} = 0$) and the bound can be written

$$\tilde{\mathcal{L}} \leq \tilde{\mathcal{L}}_1 \equiv \sum_{\ell \in P} \sum_{i=1}^{m_{\ell+1}} \sum_{j=1}^{m_\ell} \mathcal{W}_{ij}^{(\ell)}. \tag{3}$$

We call $\tilde{\mathcal{L}}_1$ the *residual partitioning bound*. We are interested in this bound because its Hessian with respect to $\{\Delta w_{ij}^{(\ell)}\}$ is diagonal, and so is easily minimized.

To prove Theorem 1, we must show how to bound through three different parts of the network: the convex loss function, the trainable affine layers, and the activation layers. However, here we only show how to bound through the affine layers to demonstrate the main technique used. The process of bounding through the other parts of the network is treated in the Appendix.

3.1. Bounding through affine layers

Affine layers include fully connected layers, convolutional layers, and residual connections. Here we only consider fully connected layers, but the work shown here generalizes to other affine layer types. For an affine layer, the i-th component of the output and change in output are

$$y_{si}^{(\ell+1)} \equiv \sum_{j=0}^{m_\ell} w_{ij}^{(\ell)} y_{sj}^{(\ell)} \qquad \Delta y_{si}^{(\ell+1)} \approx \sum_{j=0}^{m_\ell} \Delta w_{ij}^{(\ell)} y_{sj}^{(\ell)} + \sum_{j=1}^{m_\ell} w_{ij}^{(\ell)} \Delta y_{sj}^{(\ell)}. \tag{4}$$

Note that we have omitted the term $\Delta w_{ij}^{(\ell)} \Delta y_{sj}^{(\ell)}$ from $\Delta y_{si}^{(\ell+1)}$. Doing so is not necessary, but simplifies notation and the resulting algorithm. If $\ell \neq 1$, for each $s \in [S]$ and $i \in [m_{\ell+1}]$ choose partitioning variables $\{\varepsilon_{sij}^{(\ell)}\}_{j=0}^{m_\ell}$ and $\{\varphi_{sij}^{(\ell)}\}_{j=1}^{m_\ell}$ that are non-negative and together add to one:

$$\sum_{j=0}^{m_\ell} \varepsilon_{sij}^{(\ell)} + \sum_{j=1}^{m_\ell} \varphi_{sij}^{(\ell)} = 1 \quad \forall\, s, i \qquad \varepsilon_{sij}^{(\ell)} \geq 0 \quad \varphi_{sij}^{(\ell)} \geq 0 \quad \forall\, s, i, j \tag{5}$$

We can interpret $\varepsilon_{sij}^{(\ell)}$ as the fraction of the residual backpropagated through $y_{si}^{(\ell+1)}$ that will be used to update $w_{ij}^{(\ell)}$, and $\varphi_{sij}^{(\ell)}$ as the fraction that will continue to backpropagate to higher layers through $y_{sj}^{(\ell)}$. We will construct $\tilde{\mathcal{L}}_\ell$ from $\tilde{\mathcal{L}}_{\ell+1}$ by bounding the $\Delta y_{si}^{(\ell+1)2}$ term in $\mathcal{Y}_{si}^{(\ell+1)}$. To do this, multiply by one and apply Jensen's inequality:

$$\Delta y_{si}^{(\ell+1)2} = \left(\sum_{j=0}^{m_\ell} \varepsilon_{sij}^{(\ell)} \frac{\Delta w_{ij}^{(\ell)} y_{sj}^{(\ell)}}{\varepsilon_{sij}^{(\ell)}} + \sum_{j=1}^{m_\ell} \varphi_{sij}^{(\ell)} \frac{w_{ij}^{(\ell)} \Delta y_{sj}^{(\ell)}}{\varphi_{sij}^{(\ell)}} \right)^2 \tag{6}$$

$$\leq \sum_{j=0}^{m_\ell} \frac{(\Delta w_{ij}^{(\ell)} y_{sj}^{(\ell)})^2}{\varepsilon_{sij}^{(\ell)}} + \sum_{j=1}^{m_\ell} \frac{(w_{ij}^{(\ell)} \Delta y_{sj}^{(\ell)})^2}{\varphi_{sij}^{(\ell)}} \tag{7}$$

Applying this to each $\mathcal{Y}_{si}^{(\ell+1)}$ term of $\tilde{\mathcal{L}}_{\ell+1}$ yields the following bound, where $\mathcal{Y}_{sj}^{(\ell)}$ and $\mathcal{W}_{ij}^{(\ell)}$ are univariate convex quadratic functions of $\Delta y_{sj}^{(\ell)}$ and $\Delta w_{ij}^{(\ell)}$ respectively with the following coefficients:

$$\sum_{s=1}^{S}\sum_{i=1}^{m_{\ell+1}} \mathcal{Y}_{si}^{(\ell+1)} \le \sum_{i=1}^{m_{\ell+1}}\sum_{j=0}^{m_\ell} \mathcal{W}_{ij}^{(\ell)} + \sum_{s=1}^{S}\sum_{j=1}^{m_\ell} \mathcal{Y}_{sj}^{(\ell)} \tag{8}$$

$$\begin{aligned} \dot{\mathcal{W}}_{ij}^{(\ell)} &\equiv \sum_{s=1}^{S} \dot{\mathcal{Y}}_{si}^{(\ell+1)} y_{sj}^{(\ell)} & \dot{\mathcal{Y}}_{sj}^{(\ell)} &\equiv \sum_{i=1}^{m_{\ell+1}} \dot{\mathcal{Y}}_{si}^{(\ell+1)} w_{ij}^{(\ell)} \\ \ddot{\mathcal{W}}_{ij}^{(\ell)} &\equiv \sum_{s=1}^{S} \frac{\ddot{\mathcal{Y}}_{si}^{(\ell+1)} y_{sj}^{(\ell)2}}{\varepsilon_{sij}^{(\ell)}} & \ddot{\mathcal{Y}}_{sj}^{(\ell)} &\equiv \sum_{i=1}^{m_{\ell+1}} \frac{\ddot{\mathcal{Y}}_{si}^{(\ell+1)} w_{ij}^{(\ell)2}}{\varphi_{sij}^{(\ell)}} \end{aligned} \tag{9}$$

Note that $\ddot{\mathcal{Y}}_{si}^{(\ell+1)} \ge 0$ guarantees $\ddot{\mathcal{W}}_{ij}^{(\ell)} \ge 0$ and $\ddot{\mathcal{Y}}_{sj}^{(\ell)} \ge 0$, so the bound remains in the correct direction. Finally, plugging this bound (8) into $\tilde{\mathcal{L}}_{\ell+1}$ and collecting like terms yields $\tilde{\mathcal{L}}_{\ell}$.

Since residual partitioning only bounds the second-order terms of the loss function, $\dot{\mathcal{W}}_{ij}^{(\ell)}$ is simply the gradient of the loss with respect to $\Delta w_{ij}^{(\ell)}$ normally calculated by backpropagation. We say residual partitioning updates $w_{ij}^{(\ell)}$ with learning rate $1/\ddot{\mathcal{W}}_{ij}^{(\ell)}$ and from (9), we see that this learning rate is an increasing function of $\varepsilon_{sij}^{(\ell)}$. We interpret this to mean that if we assign a large fraction of the residual to $w_{ij}^{(\ell)}$, then we will update $w_{ij}^{(\ell)}$ using a large learning rate.

4. Choosing partitioning variables

Now that we have shown how to construct the residual partitioning bound $\tilde{\mathcal{L}}_1$, we need to choose values for the partitioning variables $\varepsilon_{sij}^{(\ell)}$ and $\varphi_{sij}^{(\ell)}$ that satisfy the constraint (5). There are many ways to choose the partitioning variables, but in this paper, we choose the partitioning variables so the residual partitioning bound $\tilde{\mathcal{L}}_1$ is minimized if all parameter updates are equal size, i.e., $\Delta w_{ij}^{(\ell)2}$ is constant.

Theorem 2 *If $\Delta w_{ij}^{(\ell)2}$ is constant, the choice of partitioning variables that minimizes $\tilde{\mathcal{L}}_1$ is $\varepsilon_{sij}^{(\ell)} = \varepsilon^{*(\ell)}_{sij}$ and $\varphi_{sij}^{(\ell)} = \varphi^{*(\ell)}_{sij}$:*

$$\varepsilon^{*(\ell)}_{sij} = \frac{|y_{sj}^{(\ell)}|}{u_{si}^{(\ell+1)}} \qquad \varphi^{*(\ell)}_{sij} = \frac{|w_{ij}^{(\ell)}|u_{sj}^{(\ell)}}{u_{si}^{(\ell+1)}} \qquad \textit{for } 1 < \ell < L, \quad \ell \in P \tag{10}$$

where $u_{si}^{(\ell)}$ is calculated by a forward pass through the network:

$$u_{si}^{(1)} \equiv 0 \qquad u_{si}^{(\ell+1)} \equiv \begin{cases} |\dot{\sigma}_\ell(y_{sj}^{(\ell)})|u_{sj}^{(\ell)} & \textit{for activation layers } \ell \\ \sum_{j=0}^{m_\ell} |y_{sj}^{(\ell)}| + \sum_{j=1}^{m_\ell} |w_{ij}^{(\ell)}|u_{sj}^{(\ell)} & \textit{for affine layers } \ell \end{cases} \tag{11}$$

We call $u_{si}^{(\ell)}$ the *total gradient* coming into $y_{si}^{(\ell)}$. In effect, $u_{si}^{(\ell)}$ quantifies how much $y_{si}^{(\ell)}$ can change after updating all parameters.

5. Experiments

In all of our experiments, use a mini-batch size of 100. We preprocessed all three datasets by scaling and shifting the values of both train and test data to lie in the interval $[-1, 1]$. We initialized the weights using PyTorch's default parameter initialization, which is $\mathcal{U}(-\sqrt{k}, \sqrt{k})$, where $k = 1/\texttt{in\_features}$. We use L^2 weight regularization with strength 10^{-2} for all parameters. We repeated each experiment 10 times, using random seeds 1 through 10; the results we report are the average over those 10 trials. For each experiment we plot the test and train loss every 100 epochs.

Note that one iteration of residual partitioning has the same computational complexity as SGD and Adam. In practice, we observed that when running on a CPU with 16 cores, one training step of residual partitioning took about 36% more time than one training step of Adam, and 66% more time than one training step of SGD (we used PyTorch's implementations of Adam and SGD).

In our first experiment, we train an autoencoder with residual partitioning, Adam, and SGD. The autoencoder has a $D - 64 - 2 - 64 - D$ architecture, where D is the size of the input. All layers are fully connected and use Tanh activations. The results are in Figure 1, in the Appendix. We observe that although residual partitioning does not not always achieve the best training loss, it generally overfits less compared to SGD and Adam and achieves the best validation loss on all three datasets. We speculate that residual partitioning overfits less because it computes higher quality curvature information from the local second-order approximation than Adam does from an exponential moving average of past gradients.

In our second experiment, we train a classifier network with residual partitioning, Adam, and SGD. The classifier has a $D - 400 - 100 - 36 - 10$ architecture. All layers are fully connected and use softplus activations. The results are in Figures 2 and 3, in the Appendix. Residual partitioning achieves noticeably worse cross-entropy error on all three datasets, and overfits much worse on the CIFAR-10 dataset. However, residual partitioning achieves similar test accuracy on all three datasets.

We speculate that the difference in performance of residual partitioning on the autoencoder task versus the classification task is due to the inherent difference in network architecture: if residual partitioning splits the residual amongst many parameter, Jensen's inequality may yield too loose a bound.

6. Conclusion

Adaptive gradient methods have found widespread use in the research community while interest in second-order methods has languished. In this paper, we developed a second order method that is computation efficient, memory efficient, remedies the problem of saddle points efficiently, and provides strong theoretical guarantees. Deep residual partitioning accomplishes this goal by using Jensen's inequality to construct an upper bound on the objective function with a diagonal Hessian. We demonstrate a clear advantage on autoencoder experiments, achieving better quality solutions and a reduction in overfitting. While results on classification tasks are mixed, we believe that deep residual partitioning presents a promising path to match the speed and efficiency of adaptive gradient methods but with the theoretical advantages of second order methods.

References

[1] Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2): 251–276, 1998.

[2] Sue Becker, Yann Le Cun, et al. Improving the convergence of back-propagation learning with second order methods. In *Proceedings of the 1988 connectionist models summer school*, pages 29–37, 1988.

[3] Aleksandar Botev, Hippolyt Ritter, and David Barber. Practical Gauss-Newton optimisation for deep learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 557–565. JMLR. org, 2017.

[4] Charles G Broyden. Quasi-Newton methods and their application to function minimisation. *Mathematics of Computation*, 21(99):368–381, 1967.

[5] Sheng-Wei Chen, Chun-Nan Chou, and Edward Y Chang. EA-CG: An Approximate Second-Order Method for Training Fully-Connected Neural Networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3337–3346, 2019.

[6] Felix Dangel, Philipp Hennig, and Stefan Harmeling. Modular block-diagonal curvature approximations for feedforward architectures. *arXiv preprint arXiv:1902.01813*, 2019.

[7] Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in neural information processing systems*, pages 2933–2941, 2014.

[8] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[9] Roger Grosse and Ruslan Salakhudinov. Scaling up natural gradient by sparsely factorizing the inverse fisher matrix. In *International Conference on Machine Learning*, pages 2304–2313, 2015.

[10] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[11] Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[12] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[13] Dong C Liu and Jorge Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

[14] James Martens. Deep learning via Hessian-free optimization. In *ICML*, volume 27, pages 735–742, 2010.

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

[16] James Martens and Roger Grosse. Optimizing neural networks with Kronecker-factored approximate curvature. In *International conference on machine learning*, pages 2408–2417, 2015.

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

[18] Eiji Mizutani and Stuart Dreyfus. An analysis on negative curvature induced by singularity in multi-layer neural-network learning. In *Advances in Neural Information Processing Systems*, pages 1669–1677, 2010.

[19] Razvan Pascanu and Yoshua Bengio. Revisiting natural gradient for deep networks. *arXiv preprint arXiv:1301.3584*, 2013.

[20] Barak A Pearlmutter. Fast exact multiplication by the Hessian. *Neural computation*, 6(1): 147–160, 1994.

[21] Nicolas L Roux, Pierre-Antoine Manzagol, and Yoshua Bengio. Topmoumoute online natural gradient algorithm. In *Advances in neural information processing systems*, pages 849–856, 2008.

[22] Nicol N Schraudolph. Fast curvature matrix-vector products for second-order gradient descent. *Neural computation*, 14(7):1723–1738, 2002.

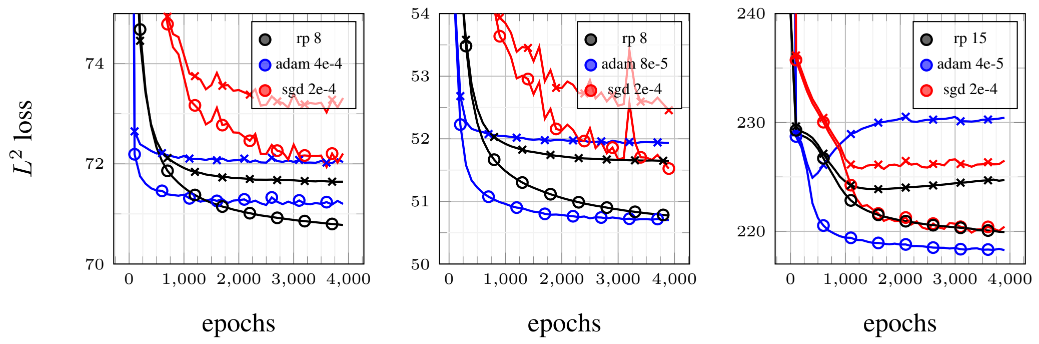
[23] Jascha Sohl-Dickstein, Ben Poole, and Surya Ganguli. Fast large-scale optimization by unifying stochastic gradient and quasi-Newton methods. In *International Conference on Machine Learning*, pages 604–612, 2014.

[24] Oriol Vinyals and Daniel Povey. Krylov subspace descent for deep learning. In *Artificial Intelligence and Statistics*, pages 1261–1268, 2012.

[25] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms, 2017.

[26] Matthew D Zeiler. ADADELTA: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

7. Appendix



(a) MNIST (b) Fashion-MNIST (c) CIFAR-10

Figure 1: L^2 loss achieved by training an autoencoder with each optimization algorithm with the specified learning rate. Train loss is marked with circles, validation loss is marked with exes. Note residual partitioning achieves the lowest validation loss in all three experiments.

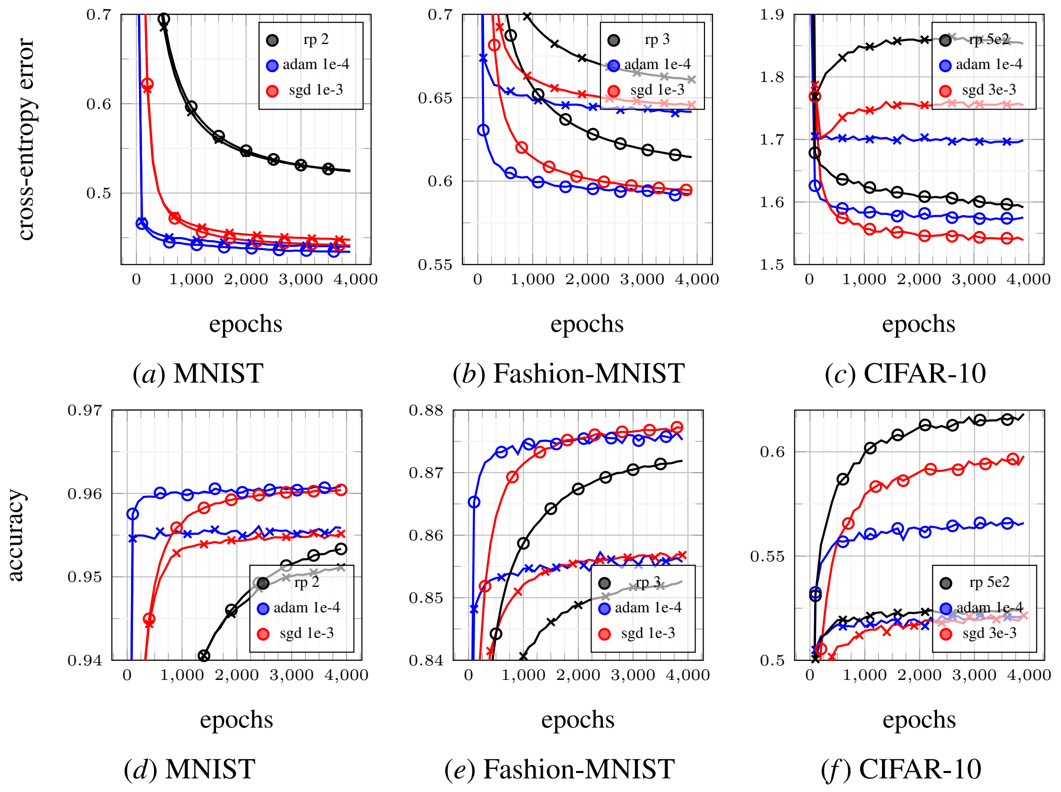


Figure 2: Cross-entropy error and accuracy achieved by training a classifier network with each optimization algorithm with the specified learning rate. Train statistics are marked with circles, validation statistics are marked with exes.

7.1. Bounding $\ell = L$

In the main body of the paper, we briefly show how to construct $\tilde{\mathcal{L}}_L$ using $\{\varphi_{si}^{(L)}\}$ when the loss function is convex and its Hessian with respect to the network outputs is not diagonal. Here is the same work, but without skipping steps. Here, $\odot$ is componentwise multiplication and $\boldsymbol{e}_i$ is the i-th standard basis vector.

$$\mathcal{L} = \sum_{s=1}^{S} f(\hat{\boldsymbol{y}}_s^{(L)}|\boldsymbol{x}_s) = \sum_{s=1}^{S} f(\boldsymbol{y}_s^{(L)} + \Delta\boldsymbol{y}_s^{(L)}|\boldsymbol{x}_s) \tag{12}$$

$$= \sum_{s=1}^{S} f\left(\boldsymbol{y}_s^{(L)} + \sum_{i=1}^{m_L} \Delta\boldsymbol{y}_s^{(L)} \odot \boldsymbol{e}_i \middle| \boldsymbol{x}_s\right) \tag{13}$$

$$= \sum_{s=1}^{S} f\left(\sum_{i=1}^{m_L} \varphi_{si}^{(L)} \left(\boldsymbol{y}_s^{(L)} + \frac{1}{\varphi_{si}^{(L)}} \Delta\boldsymbol{y}_s^{(L)} \odot \boldsymbol{e}_i\right) \middle| \boldsymbol{x}_s\right) \tag{14}$$

$$\leq \sum_{s=1}^{S} \sum_{i=1}^{m_L} \varphi_{si}^{(L)} f\left(\boldsymbol{y}_s^{(L)} + \frac{1}{\varphi_{si}^{(L)}} \Delta\boldsymbol{y}_s^{(L)} \odot \boldsymbol{e}_i \middle| \boldsymbol{x}_s\right) \tag{15}$$

7.2. Bounding $\ell = 1$

We can bound through an affine layer at the top of the network using only $\{\varepsilon_{sij}^{(\ell)}\}$, i.e., without $\{\varphi_{sij}^{(\ell))}\}$, since $\Delta y_{sj}^{(\ell)} = 0$. First, bound $\Delta y_{si}^{(\ell+1)2}$:

$$\Delta y_{si}^{(\ell+1)2} = \left(\sum_{j=0}^{m_\ell} \varepsilon_{sij}^{(\ell)} \frac{\Delta w_{ij}^{(\ell)} y_{sj}^{(\ell)}}{\varepsilon_{sij}^{(\ell)}}\right)^2 \leq \sum_{j=0}^{m_\ell} \frac{(\Delta w_{ij}^{(\ell)} y_{sj}^{(\ell)})^2}{\varepsilon_{sij}^{(\ell)}} \tag{16}$$

Applying this to each $\mathcal{Y}_{si}^{(\ell+1)}$ term of $\tilde{\mathcal{L}}_{\ell+1}$ yields

$$\sum_{s=1}^{S} \sum_{i=1}^{m_{\ell+1}} \mathcal{Y}_{si}^{(\ell+1)} \leq \sum_{i=1}^{m_{\ell+1}} \sum_{j=0}^{m_\ell} \mathcal{W}_{ij}^{(\ell)} \tag{17}$$

$$\dot{\mathcal{W}}_{ij}^{(\ell)} \equiv \sum_{s=1}^{S} \dot{\mathcal{Y}}_{si}^{(\ell+1)} y_{sj}^{(\ell)} \qquad \ddot{\mathcal{W}}_{ij}^{(\ell)} \equiv \sum_{s=1}^{S} \frac{\ddot{\mathcal{Y}}_{si}^{(\ell+1)} y_{sj}^{(\ell)2}}{\varepsilon_{sij}^{(\ell)}} \tag{18}$$

7.3. Proof of Theorem 2

Before we prove Theorem 2, let us prove a much simpler lemma that we will use extensively:

Lemma 3 *Let $\{x_i\}$ be a set of n real numbers, and let $\{p_i\}$ be a set of n non-negative real numbers that sum to one. Then the solution to the following constrained optimization problem is $p_i = p_i^*$:*

$$\min_{\{p_i\}} \sum_{i=1}^{n} \frac{x_i^2}{p_i} \qquad p_i^* \equiv \frac{|x_i|}{\sum_{i'=1}^{n} |x_{i'}|} \tag{19}$$

Proof First, let us reparameterize p_i by a set of unconstrained parameters η_i. Let $\{\eta_i\}$ be a set of n real numbers, and let

$$p_i \equiv \frac{\exp\{\eta_i\}}{\sum_{i'=1}^n \exp\{\eta_{i'}\}} \tag{20}$$

We can solve the optimization problem by setting the gradient with respect to $\{\eta_i\}$ to zero. We will need the following partial derivatives:

$$\frac{\partial p_i}{\partial \eta_i} = p_i(1-p_i) \qquad\qquad \frac{\partial p_{i'}}{\partial \eta_i} = -p_i p_{i'} \quad \text{if } i' \neq i \tag{21}$$

Now the gradient of the objective is

$$\frac{\partial}{\partial \eta_i} \sum_{i'=1}^n \frac{x_{i'}^2}{p_{i'}} = \sum_{i'=1}^n -\frac{x_{i'}^2}{p_{i'}^2}\frac{\partial p_{i'}}{\partial \eta_i} = p_i\left(\frac{x_i^2}{p_i^2} - \sum_{i'=1}^n p_{i'}\frac{x_{i'}^2}{p_{i'}^2}\right) \tag{22}$$

The gradient is zero in particular when x_i^2/p_i^2 is constant across i: if $x_i^2/p_i^2 = c$, then

$$p_i\left(\frac{x_i^2}{p_i^2} - \sum_{i'=1}^n p_{i'}\frac{x_{i'}^2}{p_{i'}^2}\right) = p_i\left(c - \sum_{i'=1}^n p_{i'}c\right) = p_i(c-c) = 0 \tag{23}$$

Note x_i^2/p_i^2 is constant across i in particular when $\eta_i = \log|x_i|$, so $p_i = |x_i|/\sum_{i'=1}^n |x_{i'}|$ and

$$\frac{x_i^2}{p_i^2} = \frac{x_i^2}{(|x_i|/\sum_{i'=1}^n |x_{i'}|)^2} = \left(\sum_{i'=1}^n |x_{i'}|\right)^2 = c. \tag{24}$$

■

Now let us use the lemma to prove the theorem:

Theorem 2 *If $\Delta w_{ij}^{(\ell)2}$ is constant, the choice of partitioning variables that minimizes $\tilde{\mathcal{L}}_1$ is $\varepsilon_{sij}^{(\ell)} = \varepsilon^{*(\ell)}_{sij}$ and $\varphi_{sij}^{(\ell)} = \varphi^{*(\ell)}_{sij}$:*

$$\varepsilon^{*(\ell)}_{sij} = \frac{|y_{sj}^{(\ell)}|}{u_{si}^{(\ell+1)}} \qquad \varphi^{*(\ell)}_{sij} = \frac{|w_{ij}^{(\ell)}|u_{sj}^{(\ell)}}{u_{si}^{(\ell+1)}} \qquad \textit{for } 1<\ell<L, \quad \ell \in P \tag{10}$$

where $u_{si}^{(\ell)}$ is calculated by a forward pass through the network:

$$u_{si}^{(1)} \equiv 0 \qquad u_{si}^{(\ell+1)} \equiv \left\{ \begin{array}{cc} |\dot{\sigma}_\ell(y_{sj}^{(\ell)})|u_{sj}^{(\ell)} & \textit{for activation layers } \ell \\ \sum_{j=0}^{m_\ell}|y_{sj}^{(\ell)}| + \sum_{j=1}^{m_\ell}|w_{ij}^{(\ell)}|u_{sj}^{(\ell)} & \textit{for affine layers } \ell \end{array}\right. \tag{11}$$

Proof We will optimize over the partitioning variables one layer at a time working forwards. More formally, we will prove the theorem by induction. To do this, we need to strengthen the inductive hypothesis by also proving the following holds for all $\ell \in \{0\} \cup [L-1]$, where *const* is a constant with respect to all partitioning variables:

$$\sum_{\ell' \in P, \ell' \leq \ell} \sum_{i=1}^{m_{\ell'+1}} \sum_{j=0}^{m_{\ell'}} \ddot{\mathcal{W}}_{ij}^{(\ell')} = \sum_{s=1}^{S} \sum_{i=1}^{m_{\ell+1}} \ddot{\mathcal{Y}}_{si}^{(\ell+1)} u_{si}^{(\ell+1)2} + const \tag{25}$$

The base case $\ell = 0$ is simple: the left hand side of (25) is zero since there are no terms to sum over, and the right hand side is zero since $u_{si}^{(1)} \equiv 0$. Now assume the inductive hypothesis (25) is true for $\ell - 1$, and let us prove it holds for ℓ.

Suppose layer ℓ is an affine layer. Then the left hand side of (25) has one additional term for ℓ compared to $\ell - 1$, which we can pull out before applying the inductive hypothesis (25):

$$\sum_{\ell' \in P, \ell' \leq \ell} \sum_{i=1}^{m_{\ell'+1}} \sum_{j=0}^{m_{\ell'}} \ddot{\mathcal{W}}_{ij}^{(\ell')} = \sum_{\ell' \in P, \ell' \leq \ell-1} \sum_{i=1}^{m_{\ell'+1}} \sum_{j=0}^{m_{\ell'}} \ddot{\mathcal{W}}_{ij}^{(\ell')} + \sum_{i=1}^{m_{\ell+1}} \sum_{j=0}^{m_{\ell}} \ddot{\mathcal{W}}_{ij}^{(\ell)} \tag{26}$$

$$= \sum_{s=1}^{S} \sum_{j=1}^{m_\ell} \ddot{\mathcal{Y}}_{sj}^{(\ell)} u_{sj}^{(\ell)2} + \sum_{i=1}^{m_{\ell+1}} \sum_{j=0}^{m_{\ell}} \ddot{\mathcal{W}}_{ij}^{(\ell)} + const \tag{27}$$

If we are at the top of the network, i.e., $\ell = 1$, then $u_{sj}^{(\ell)} \equiv 0$ and plugging in the formula for $\ddot{\mathcal{W}}_{ij}^{(\ell)}$ (18) yields

$$\sum_{s=1}^{S} \sum_{j=1}^{m_\ell} \ddot{\mathcal{Y}}_{sj}^{(\ell)} u_{sj}^{(\ell)2} + \sum_{i=1}^{m_{\ell+1}} \sum_{j=0}^{m_{\ell}} \ddot{\mathcal{W}}_{ij}^{(\ell)} \tag{28}$$

$$= \sum_{i=1}^{m_{\ell+1}} \sum_{j=0}^{m_{\ell}} \ddot{\mathcal{W}}_{ij}^{(\ell)} \tag{29}$$

$$= \sum_{s=1}^{S} \sum_{i=1}^{m_{\ell+1}} \ddot{\mathcal{Y}}_{si}^{(\ell+1)} \sum_{j=0}^{m_\ell} \frac{y_{sj}^{(\ell)2}}{\varepsilon_{sij}^{(\ell)}} \tag{30}$$

Since for each $s \in [S]$ and $i \in [m_{\ell+1}]$ we have $\{\varepsilon_{sij}^{(\ell)}\}$ are non-negative and sum to one, they fulfill the role of $\{p_i\}$ in our lemma. Applying the lemma, the optimal choice of $\{\varepsilon_{sij}^{(\ell)}\}$ is

$$\varepsilon^{*(\ell)}_{sij} \equiv \frac{|y_{sj}^{(\ell)}|}{\sum_{j'=1}^{m_\ell} |y_{sj'}^{(\ell)}|} \tag{31}$$

Using the formula for $u_{si}^{(\ell+1)}$ (11) and $u_{sj}^{(\ell)} \equiv 0$ (since $\ell = 1$), we can write this as

$$\varepsilon^{*(\ell)}_{sij} \equiv \frac{|y_{sj}^{(\ell)}|}{u_{si}^{(\ell+1)}} \tag{32}$$

Plugging this back into (30) yields

$$\sum_{s=1}^{S} \sum_{i=1}^{m_{\ell+1}} \ddot{\mathcal{Y}}_{si}^{(\ell+1)} \sum_{j=0}^{m_\ell} \frac{y_{sj}^{(\ell)2}}{\varepsilon_{sij}^{(\ell)}} = \sum_{s=1}^{S} \sum_{i=1}^{m_{\ell+1}} \ddot{\mathcal{Y}}_{si}^{(\ell+1)} u_{si}^{(\ell+1)2} \tag{33}$$

and the inductive hypothesis holds. If layer ℓ is affine but we are not at the top of the network, i.e., $\ell \neq 1$, plugging the formulae for $\ddot{\mathcal{Y}}_{sj}^{(\ell)}$ and $\ddot{\mathcal{W}}_{ij}^{(\ell)}$ (9) into (27) yields

$$\sum_{s=1}^{S}\sum_{j=1}^{m_\ell}\ddot{\mathcal{Y}}_{sj}^{(\ell)}u_{sj}^{(\ell)2}+\sum_{i=1}^{m_{\ell+1}}\sum_{j=0}^{m_\ell}\ddot{\mathcal{W}}_{ij}^{(\ell)} \tag{34}$$

$$=\sum_{s=1}^{S}\sum_{i=1}^{m_{\ell+1}}\sum_{j=1}^{m_\ell}\ddot{\mathcal{Y}}_{si}^{(\ell+1)}\frac{w_{ij}^{(\ell)2}}{\varphi_{sij}^{(\ell)}}u_{sj}^{(\ell)2}+\sum_{s=1}^{S}\sum_{i=1}^{m_{\ell+1}}\sum_{j=0}^{m_\ell}\ddot{\mathcal{Y}}_{si}^{(\ell+1)}\frac{y_{sj}^{(\ell)2}}{\varepsilon_{sij}^{(\ell)}} \tag{35}$$

$$=\sum_{s=1}^{S}\sum_{i=1}^{m_{\ell+1}}\ddot{\mathcal{Y}}_{si}^{(\ell+1)}\left(\sum_{j=1}^{m_\ell}\frac{\left(w_{ij}^{(\ell)}u_{sj}^{(\ell)}\right)^2}{\varphi_{sij}^{(\ell)}}+\sum_{j=0}^{m_\ell}\frac{y_{sj}^{(\ell)2}}{\varepsilon_{sij}^{(\ell)}}\right) \tag{36}$$

Since for each $s \in [S]$ and $i \in [m_{\ell+1}]$ we have $\{\varepsilon_{sij}^{(\ell)}\}$ and $\{\varphi_{sij}^{(\ell)}\}$ are non-negative and together sum to one, they fulfill the role of $\{p_i\}$ in our lemma. Applying the lemma, the optimal choice of $\{\varepsilon_{sij}^{(\ell)}\}$ and $\{\varphi_{sij}^{(\ell)}\}$ is

$$\varepsilon^{*(\ell)}_{sij}\equiv\frac{|y_{sj}^{(\ell)}|}{\sum_{j'=0}^{m_\ell}|y_{sj'}^{(\ell)}|+\sum_{j'=1}^{m_\ell}|w_{ij'}^{(\ell)}|u_{sj'}^{(\ell)}}\qquad\varphi^{*(\ell)}_{sij}\equiv\frac{|w_{ij}^{(\ell)}|u_{sj}^{(\ell)}}{\sum_{j'=0}^{m_\ell}|y_{sj'}^{(\ell)}|+\sum_{j'=1}^{m_\ell}|w_{ij'}^{(\ell)}|u_{sj'}^{(\ell)}} \tag{37}$$

Using the formula for $u_{si}^{(\ell+1)}$ (11) we can write this as

$$\varepsilon^{*(\ell)}_{sij}\equiv\frac{|y_{sj}^{(\ell)}|}{u_{si}^{(\ell+1)}}\qquad\varphi^{*(\ell)}_{sij}\equiv\frac{|w_{ij}^{(\ell)}|u_{sj}^{(\ell)}}{u_{si}^{(\ell+1)}} \tag{38}$$

Plugging this back into (36) yields

$$\sum_{s=1}^{S}\sum_{i=1}^{m_{\ell+1}}\ddot{\mathcal{Y}}_{si}^{(\ell+1)}\left(\sum_{j=1}^{m_\ell}\frac{\left(w_{ij}^{(\ell)}u_{sj}^{(\ell)}\right)^2}{\varphi_{sij}^{(\ell)}}+\sum_{j=0}^{m_\ell}\frac{y_{sj}^{(\ell)2}}{\varepsilon_{sij}^{(\ell)}}\right)=\sum_{s=1}^{S}\sum_{i=1}^{m_{\ell+1}}\ddot{\mathcal{Y}}_{si}^{(\ell+1)}u_{si}^{(\ell+1)2} \tag{39}$$

and the inductive hypothesis (25) holds.

If layer ℓ is an activation layer, then there are no additional terms on the left hand side for ℓ compared to $\ell-1$, so

$$\sum_{\ell'\in P,\ell'\leq\ell}\sum_{i=1}^{m_{\ell'+1}}\sum_{j=0}^{m_{\ell'}}\ddot{\mathcal{W}}_{ij}^{(\ell')}=\sum_{\ell'\in P,\ell'\leq\ell-1}\sum_{i=1}^{m_{\ell'+1}}\sum_{j=0}^{m_{\ell'}}\ddot{\mathcal{W}}_{ij}^{(\ell')} \tag{40}$$

For the right hand side, we plug in the formula for $\ddot{\mathcal{Y}}_{sj}^{(\ell)}$ and the formula for $u_{si}^{(\ell+1)}$ (11) to get

$$\sum_{s=1}^{S}\sum_{j=1}^{m_\ell}\ddot{\mathcal{Y}}_{sj}^{(\ell)}u_{sj}^{(\ell)2}=\sum_{s=1}^{S}\sum_{i=1}^{m_{\ell+1}}\left(\ddot{\mathcal{Y}}_{si}^{(\ell+1)}\dot{\sigma}_\ell(y_{sj}^{(\ell)})^2+\dot{\mathcal{Y}}_{si}^{(\ell+1)}\ddot{\sigma}(y_{sj}^{(\ell)})\right)u_{sj}^{(\ell)2} \tag{41}$$

$$=\sum_{s=1}^{S}\sum_{i=1}^{m_{\ell+1}}\ddot{\mathcal{Y}}_{si}^{(\ell+1)}\dot{\sigma}_\ell(y_{sj}^{(\ell)})^2u_{sj}^{(\ell)2}+\mathit{const} \tag{42}$$

$$=\sum_{s=1}^{S}\sum_{i=1}^{m_{\ell+1}}\ddot{\mathcal{Y}}_{si}^{(\ell+1)}u_{si}^{(\ell+1)2}+\mathit{const} \tag{43}$$

Putting (40) and (43) together gives the inductive hypothesis (25). There are no partitioning variables to optimize over in an activation layer, so we are done with this case.

Finally, if $\ell = L$, the inductive hypothesis tells us

$$\tilde{\mathcal{L}}_1 = \sum_{\ell \in P} \sum_{i=1}^{m_{\ell'+1}} \sum_{j=0}^{m_{\ell'}} \ddot{\mathcal{W}}_{ij}^{(\ell')} = \sum_{s=1}^{S} \sum_{i=1}^{m_L} \ddot{\mathcal{Y}}_{si}^{(L)} u_{si}^{(L)2} \tag{44}$$

Plugging in the formula for $\ddot{\mathcal{Y}}_{si}^{(L)}$ yields

$$\sum_{s=1}^{S} \sum_{i=1}^{m_L} \ddot{\mathcal{Y}}_{si}^{(L)} u_{si}^{(L)2} = \sum_{s=1}^{S} \sum_{i=1}^{m_L} \frac{1}{\varphi_{si}^{(L)}} \frac{\partial^2 f(\boldsymbol{y}_s^{(L)} | \boldsymbol{x}_s)}{\partial y_{si}^{(L)2}} u_{si}^{(L)2} \tag{45}$$

Since for each $s \in [S]$ we have $\{\varphi_{si}^{(\ell)}\}$ are non-negative and sum to one, they fulfill the role of $\{p_i\}$ in our lemma. Applying the lemma, the optimal choice of $\{\varphi_{si}^{(L)}\}$ is

$$\varphi^{*(\ell)}_{si} \propto |u_{si}^{(L)}| \sqrt{\frac{\partial^2 f(\boldsymbol{y}_s^{(L)} | \boldsymbol{x}_s)}{\partial y_{si}^{(L)2}}} \tag{46}$$

■

Algorithm 1 One training step of residual partitioning.

```
INPUT: Network inputs $\{\boldsymbol{y}_s^{(1)}\}$.
INPUT: Network targets $\{\boldsymbol{x}_s\}$.
INPUT: Current weights $\{w_{ij}^{(\ell)}\}$.
INPUT: Global learning rate $\lambda > 0$.
OUTPUT: Weight updates $\{\Delta w_{ij}^{(\ell)}\}$.

for $s \in [S]$, $i \in [m_1]$ do // Initialize Foward Pass
    $u_{si}^{(0)} \leftarrow 0$
end for
for $\ell \leftarrow 1, \ldots, L-1$ do // Foward Pass
    for $i \in [m_{\ell+1}]$ do
        if layer $l$ is an affine layer then
            $u_{si}^{(\ell+1)} \leftarrow \sum_{j=0}^{m_\ell} |y_{sj}^{(\ell)}| + \sum_{j=1}^{m_\ell} |w_{ij}^{(\ell)}| u_{sj}^{(\ell)}$
            $y_{si}^{(\ell+1)} \leftarrow \sum_{j=0}^{m_\ell} w_{ij}^{(\ell)} y_{sj}^{(\ell)}$
        else if layer $l$ is an activation layer then
            $u_{si}^{(\ell+1)} \leftarrow |\dot{\sigma}_\ell(y_{sj}^{(\ell)})| u_{sj}^{(\ell)}$
            $y_{si}^{(\ell+1)} \leftarrow \sigma(y_{sj}^{(\ell)})$
        end if
    end for
end for
for $s \in [S]$, $i \in [m_L]$ do // Initialize Backward Pass
    $\dot{\mathcal{Y}}_{si}^{(L)} \leftarrow \frac{\partial f(\boldsymbol{y}_s^{(L)}|\boldsymbol{x}_s)}{\partial y_{si}^{(L)}}$
    $\ddot{\mathcal{Y}}_{si}^{(L)} u_{si}^{(L)} \leftarrow \sqrt{\frac{\partial^2 f(\boldsymbol{y}_s^{(L)}|\boldsymbol{x}_s)}{\partial y_{si}^{(L)2}}} \sum_{i'=1}^{m_L} |u_{si'}^{(L)}| \sqrt{\frac{\partial^2 f(\boldsymbol{y}_s^{(L)}|\boldsymbol{x}_s)}{\partial y_{si'}^{(L)2}}}$
end for
for $\ell \leftarrow L-1, \ldots, 1$ do // Backward Pass
    if layer $l$ is an affine layer then
        for $i \in [m_{\ell+1}]$, $j \in [m_\ell]$ do
            $\dot{\mathcal{W}}_{ij}^{(\ell)} \leftarrow \frac{1}{S} \sum_{s=1}^{S} \dot{\mathcal{Y}}_{si}^{(\ell+1)} y_{sj}^{(\ell)}$
            $\ddot{\mathcal{W}}_{ij}^{(\ell)} \leftarrow \frac{1}{S} \sum_{s=1}^{S} \ddot{\mathcal{Y}}_{si}^{(\ell+1)} u_{si}^{(\ell+1)} |y_{sj}^{(\ell)}|$
        end for
        for $s \in [S]$, $j \in [m_\ell]$ do
            $\dot{\mathcal{Y}}_{sj}^{(\ell)} \leftarrow \sum_{i=1}^{m_{\ell+1}} \dot{\mathcal{Y}}_{si}^{(\ell+1)} w_{ij}^{(\ell)}$
            $\ddot{\mathcal{Y}}_{sj}^{(\ell)} \leftarrow \sum_{i=1}^{m_{\ell+1}} \ddot{\mathcal{Y}}_{si}^{(\ell+1)} u_{si}^{(\ell+1)} |w_{ij}^{(\ell)}|$
        end for
    else if layer $l$ is an activation layer then
        for $s \in [S]$, $j \in [m_\ell]$ do
            $\dot{\mathcal{Y}}_{sj}^{(\ell)} \leftarrow \dot{\mathcal{Y}}_{si}^{(\ell+1)} \dot{\sigma}_\ell(y_{sj}^{(\ell)})$
            $\ddot{\mathcal{Y}}_{sj}^{(\ell)} \leftarrow |\ddot{\mathcal{Y}}_{si}^{(\ell+1)} \dot{\sigma}_\ell(y_{sj}^{(\ell)})^2 + \dot{\mathcal{Y}}_{si}^{(\ell+1)} \ddot{\sigma}_\ell(y_{sj}^{(\ell)})|$
        end for
    end if
end for
for $\ell \in P$, $i \in [m_{\ell+1}]$, $j \in \{0\} \cup [m_\ell]$ do // Return
    $\Delta w_{ij}^{(\ell)} \leftarrow -\lambda \cdot \dot{\mathcal{W}}_{ij}^{(\ell)} / \ddot{\mathcal{W}}_{ij}^{(\ell)}$
end for
return $\{\Delta w_{ij}^{(\ell)}\}$
```