Robust Coding Strategies for Neural Network Weights (Theory)

Berivan Isik*
berivan0
Department of Electrical Engineering
Stanford University
berivan.isik@stanford.edu

1 Introduction

The success of deep neural networks (NNs) in recent years has come with their growing size and overparameterization [1, 2]. This makes it critical to efficiently compress, store and communicate the NN weights [3]. Although NN compression has attracted great attention, and a wide variety of methods have been developed such as pruning, quantization and knowledge distillation [4, 5, 6, 7, 8], there has been significantly less interest in the efficient storage and communication of NN weights. The reason why efficient storage has not been seen as an interesting problem is that NN weights are conventionally being stored as binary arrays, which can be protected by error correction codes (ECC) developed over the past 50 years [9]. However, recent studies have demonstrated the promise of end-to-end analog memory systems for storing NN weights, as they have the potential to reach higher storage capacities than binary systems with a considerably lower coding complexity [10, 11, 12]. Although analog storage devices have these advantages over binary devices, their noisy characteristics makes it necessary to apply analog error correction codes before storage. Similarly, efficient communication has not attracted enough attention, because NN weights are usually digitally communicated using conventional error correction codes [9] in various applications such as federated learning. The error correction schemes that protects NN weights against the noise in analog storage systems, can also provide reliable communication of NN weights through wireless channels. Thus by developing robust coding strategies against noise added to the NN weights, we can provide 1) analog storage of NN weights in full-precision, 2) reliable wireless communication of NN weights.

Motivated by the discussion above, I have investigated the performance degradation of NNs when their weights are noisy, and developed robust coding strategies to preserve the performance of NNs. The source code is available in `https://github.com/BerivanIsik/CS_230_project`.

2 Related work

Even though it has not attracted as much interest as NN compression, there has been some practical work on making NNs more robust to noise. [13] trained a student network more robust to Gaussian noise by injecting noise during distillation from the teacher. [14, 15] developed error correction codes against Gaussian noise in analog storage and bit flips in binary storage.

3 Method

In a noisy storage or communication channel, the value of the input must lie within a range due to power constraints. In this project, I choose this range as $[-1, 1]$. This means that we need to scale the

*`https://sites.google.com/view/berivanisik`

weights into $[-1, 1]$ before noise addition. Then the relationship between true weights w_{in} and noisy weights w_{out} can be written as:

$$w_{out} = \frac{\phi(\alpha w_{in} - \beta) + \beta}{\alpha} \tag{1}$$

where ϕ represents the noisy medium, and α, β are scale and shift factors, respectively. The noisy medium ϕ can be modelled as an additive noise channel and we can rewrite the relationship as:

$$w_{out} = \frac{\big((\alpha w_{in} - \beta) + \epsilon\big) + \beta}{\alpha} = w_{in} + \frac{\epsilon}{\alpha} \tag{2}$$

where ϵ is the additive channel noise. When I pick a suitable α to map weights into $[-1, 1]$ range without applying any protection scheme, the accuracy dropped to 10% (random prediction for CIFAR-10) for noise standard deviation (std) larger than 0.001 (see Table 1, Figure 4.2). A naive way of boosting the accuracy is to use the channel multiple times per weight. This means that, depending on the problem, we can store the same weight on multiple memory cells or communicate the same weight multiple times, and average over the read/received values. However this brings the problem of redundant resource usage. In the next section, I will describe several practical coding strategies to increase the accuracy and shift the accuracy drop threshold to std$= 1$ (1000 times larger than the threshold when there is no protection) without needing any redundancy, i.e. 1 channel use per weight.

3.1 Practical Coding Strategy

Sign Protection: When scaling the weights by α to fit them in range $[-1, 1]$ of ϕ, small weights are mapped to values very close to zero. This is problematic because a majority of the trained NN weights have small magnitudes, and thus the NN with reconstructed weights will suffer a severe drop in performance due to sign errors (see Figure 3.1). When we store/communicate magnitudes instead of actual weights, we can use an α that is twice as large, reducing the variance of noise because $\frac{\epsilon}{\alpha}$ in Equation 2 has smaller variance when α is larger.

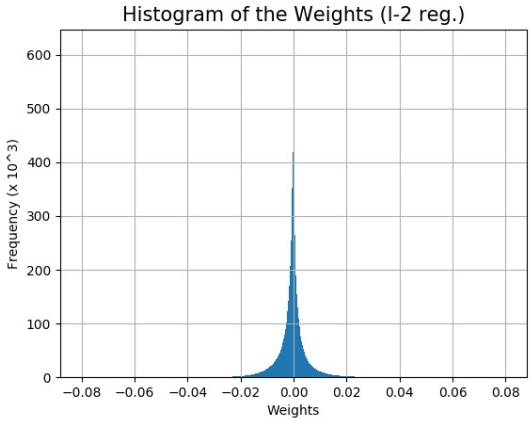

Figure 1: Histogram of ResNet-18 weights trained on CIFAR10.

Adaptive Mapping: Although protecting the sign bit leads to accuracy gains (see Table 1), we can still do better. Based on the observation that the majority of the weights are quite small (see Figure 3.1), an adaptive mapping strategy would help increase the accuracy by protecting the small weights. This implies that using different values of α depending on the magnitude of the weight (larger α for small weights) can reduce the overall variance of the noise added to small weights.

Adaptive Redundancy: As the last step, I propose to vary the number of channel usage for larger and smaller weights. The average number of channel usage that we aim to minimize is:

$$r_{avg} = \frac{r_{small} \times N_{small} + r_{large} \times N_{large}}{N_{small} + N_{large}}.$$

where N_{small} and N_{large} are the number of small and large weights; r_{small} and r_{large} are the number of repetitions used per weight for small and large weights. Adding more repetitions for larger weights (which are more critical for NN performance) increases the accuracy while it does not increase the average redundancy too much. As shown in Figure 3.1, most weights are sparse, e.g. $N_{small} = 11,157,354.$ and $N_{large} = 6,998$ in ResNet-18 trained on CIFAR-10.

Adaptive mapping and adaptive redundancy are similar to the adaptive coding strategies developed for task specific error correction codes (ECC). For instance, small and large inputs are treated differently for product operations in [16].

3.2 Robustness through Regularization

In the previous section, I presented sign protection and adaptive mapping strategies that protect small weights and this helped us get better accuracy because of the sparsity of the trained NN weights. We can get even better results using these practical coding strategies if we have more sparse NN weights. In order to see the effect of sparsity in the success of these strategies, I repeat the experiments in 4 different settings:

1) No regularization, no dropout (in Table 2),
2) $l-1$ regularization, no dropout (in Table 3),
3) $l-2$ regularization, no dropout (in Table 1, Figure 4.2),
4) no regularization, dropout (in Table 4).

We would expect the networks trained with $l-1$ regularization and dropout to be more sparse and hence, together with the strategies in Section 3.1, more robust to noise added on the weights.

4 Experiments

4.1 Datasets, Models and Hyperparameters

For the experiments, I consider CIFAR10 [17] dataset. I use the standard train/val/test splits for CIFAR10 and apply random horizontal flips as data augmentation during training. I use ResNet-18 [18]. I use a batch size of 128 and train for 350 epochs, early stopping at the best accuracy. For all the settings, I use the same learning rate $= 0.1$ as it gives the best result among $\{0.1, 0.01, 0.001\}$. I use the conventional default momentum parameter $\beta = 0.9$ in all experiments. In the $l-1$ and $l-2$ regularization experiments, I use weight decays $= 5e^{-6}$ and $= 5e^{-4}$, respectively. $5e^{-4}$ is a default value for the weight decay in $l-2$ regularization. I choose $5e^{-6}$ in $l-1$ regularization as it gives the best result among $\{5e^{-4}, 5e^{-5}, 5e^{-6}, 5e^{-7}\}$. In the dropout experiment, I put dropout layer with $p = 0.8$ after each layer in ResNet-18. For the hyperparameters I tune, I use the cavier strategy by running the models in parallel.

4.2 Results

Figure 4.2 shows that the adaptive protection strategy developed in this work improves NN performance against additive white Gaussian noise on the weights. The performance improvement gained from each strategy can be seen in Table 1, where I apply additive white Gaussian noise to the weights first without any protection, then add sign protection, adaptive mapping, and adaptive redundancy. When I apply adaptive redundancy, I kept average number of channel usage per weight less than $r_{avg} = 1.006$ by choosing $N_{small} = 11,157,354$, $N_{large} = 6998$, $r_{small} = 1$, $r_{large} = 10$. Accuracy drops to 9.547% (random prediction) when Gaussian noise with 0.05 standard deviation is added without protection (without sign protection, adaptive mapping and adaptive redundancy). When our adaptive protection strategies are applied, accuracy (95.26%) is comparable to the baseline accuracy (95.45%) even with standard deviation 0.1.

As can be seen in Tables 1, 2, 3 and 4, adaptive protection strategy improves NN performance against noise in all settings where NNs are trained with $l-2$ regularization, no regularization, $l-1$ regularization and dropout. In all the experiments, the average number of channel usage per weight

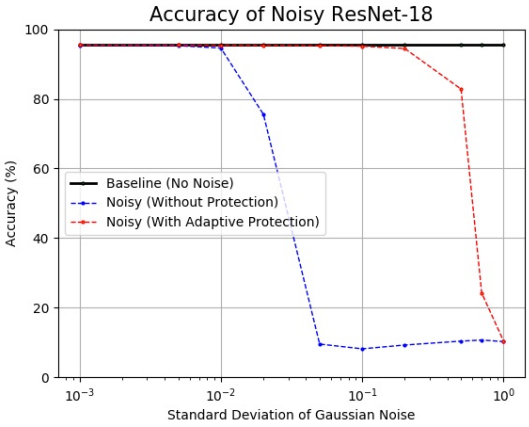

Figure 2: Accuracy of ResNet-18 on CIFAR10 (trained with l-2 regularization) when weights are perturbed by Gaussian noise.

	std=0.001	std=0.005	std=0.01	std=0.05	std=0.1	std=0.5	std=1
Test acc. (+noise) without protection	95.40	95.27	94.60	9.54	8.18	10.39	10.26
Test acc. (+noise) +sign protection	95.42	95.25	95.03	74.49	10.14	10.02	10.01
Test acc. (+noise) +adaptive mapping	95.42	95.40	95.40	95.08	94.00	15.97	10.07
Test acc. (+noise) +adaptive redundancy	95.45	95.43	95.42	95.26	95.12	82.81	10.39

Table 1: Accuracy of ResNet-18 on CIFAR10 (trained with l-2 regularization, no dropout) when weights are perturbed by Gaussian noise. The baseline test accuracy without any added noise is 95.45. The average number of channel usage per weight is kept below 1.006 by choosing $N_{small} = 11, 157, 354$, $N_{large} = 6998$, $r_{small} = 1$, $r_{large} = 10$.

	std=0.001	std=0.005	std=0.01	std=0.05	std=0.1	std=0.5	std=1
Test acc. (+noise) without protection	93.04	92.74	90.81	10.01	10.33	9.45	9.15
Test acc. (+noise) +adaptive protection	93.17	93.15	93.18	93.13	92.96	73.28	10.69

Table 2: Accuracy of ResNet-18 on CIFAR10 when weights (no regularization, no dropout) are perturbed by Gaussian noise. The baseline test accuracy without any added noise is 93.17. The average number of channel usage per weight is kept below 1.006.

	std=0.001	std=0.005	std=0.01	std=0.05	std=0.1	std=0.5	std=1
Test acc. (+noise) without protection	93.20	91.88	88.72	9.50	10.00	9.91	10.06
Test acc. (+noise) +adaptive protection	93.32	93.29	93.32	93.28	93.01	84.55	10.05

Table 3: Accuracy of ResNet-18 on CIFAR10 when weights (trained with l-1 regularization) are perturbed by Gaussian noise. The baseline test accuracy without any added noise is 93.30. The average number of channel usage per weight is kept below 1.006.

	std=0.001	std=0.005	std=0.01	std=0.05	std=0.1	std=0.5	std=1
Test acc. (+noise) without protection	93.16	92.75	91.05	10.03	10.08	11.75	11.34
Test acc. (+noise) +adaptive redundancy	93.13	93.15	93.14	93.13	92.98	81.96	10.37

Table 4: Accuracy of ResNet-18 on CIFAR10 when weights (trained with dropout) are perturbed by Gaussian noise. The baseline test accuracy without any added noise is 93.15. The average number of channel usage per weight is kept below 1.006.

is kept below 1.006 by choosing $r_{small} = 1$, $r_{large} = 10$, and N_{small} and N_{large} appropriately. It is not possible to directly compare the success of the adaptive protection on these different settings because the baseline accuracy (test accuracy without noise) is different for each setting. However we can still compare the respective accuracy drop after the noise addition as in Table 5. It is seen from Table 5 that the smallest accuracy drop occurs in either NN trained with $l-1$ regularization or dropout in all noise levels. This is consistent with the intuition that adaptive protection strategies work better as the NN gets more sparse.

		std=0.001	std=0.005	std=0.01	std=0.05	std=0.1	std=0.5	std=1
No regularization	Accuracy drop (no protection)	0.13	0.43	2.36	83.16	82.84	83.72	84.02
	Accuracy drop (+adaptive protection)	0.00	0.02	-0.01	0.04	0.21	19.89	82.48
l-1 regularization	Accuracy drop (no protection)	0.10	1.42	4.58	83.8	83.3	83.39	83.24
	Accuracy drop (+adaptive protection)	**-0.02**	0.01	**-0.02**	**0.02**	0.29	**8.75**	83.25
l-2 regularization	Accuracy drop (no protection)	0.05	0.18	0.85	83.91	87.27	85.06	85.19
	Accuracy drop (+adaptive protection)	0.00	0.02	0.03	0.19	0.33	12.64	85.06
Dropout	Accuracy drop (no protection)	-0.01	0.40	2.10	83.12	83.07	81.4	**81.81**
	Accuracy drop (+adaptive protection)	0.02	**0.00**	0.01	**0.02**	**0.17**	11.19	82.78

Table 5: Accuracy drop of ResNet-18 on CIFAR10 when weights are perturbed by Gaussian noise. The baseline test accuracy is different for each setting. The average number of channel usage per weight is kept below 1.006.

4.3 Conclusion

In this project, I developed robust coding strategies (called adaptive protection) to increase accuracy in the presence of noise added to weights. These strategies can be helpful in various applications such as analog storage and wireless communication of NN weights. I observed that these strategies make sparse NNs, trained with $l-1$ regularization and dropout, more robust to additive white Gaussian noise. This is not a surprising finding as the success of adaptive protection relies on the fact that a fully-trained NN is sparse.

4.4 Acknowledgment

I would like to thank Tsachy Weissman, Armin Alaghi, Kristy Choi, Xin Zheng, Stefano Ermon and Philip Wong for their contributions in a parallel work that led to the analysis presented in this report.

References

[1] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231, 2012.

[2] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[3] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282*, 2017.

[4] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *International Conference on Learning Representations (ICLR)*, 2016.

[5] Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*, 2015.

[6] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *International Conference on Learning Representations (ICLR)*, 2019.

[7] Weihao Gao, Yu-Han Liu, Chong Wang, and Sewoong Oh. Rate distortion for model compression: From theory to practice. In *International Conference on Machine Learning*, pages 2102–2111. PMLR, 2019.

[8] S. Wiedemann, H. Kirchhoffer, S. Matlage, P. Haase, A. Marban, T. Marinč, D. Neumann, T. Nguyen, H. Schwarz, T. Wiegand, D. Marpe, and W. Samek. Deepcabac: A universal compression algorithm for deep neural networks. *IEEE Journal of Selected Topics in Signal Processing*, 14(4):700–714, 2020.

[9] Claude Elwood Shannon. A mathematical theory of communication. *ACM SIGMOBILE mobile computing and communications review*, 5(1):3–55, 2001.

[10] Ryan Zarcone, Dylan Paiton, Alex Anderson, Jesse Engel, HS Philip Wong, and Bruno Olshausen. Joint source-channel coding with neural networks for analog data compression and storage. In *2018 Data Compression Conference*, pages 147–156. IEEE, 2018.

[11] Xin Zheng, Ryan Zarcone, Dylan Paiton, Joon Sohn, Weier Wan, Bruno Olshausen, and H-S Philip Wong. Error-resilient analog image storage and compression with analog-valued rram arrays: an adaptive joint source-channel coding approach. In *2018 IEEE International Electron Devices Meeting (IEDM)*, pages 3–5. IEEE, 2018.

[12] Ryan V Zarcone, Jesse H Engel, S Burc Eryilmaz, Weier Wan, SangBum Kim, Matthew BrightSky, Chung Lam, Hsiang-Lan Lung, Bruno A Olshausen, and H-S Philip Wong. Author correction: Analog coding in emerging memory systems. *Scientific reports*, 10(1):13404, August 2020.

[13] Chuteng Zhou, Prad Kadambi, Matthew Mattina, and Paul N Whatmough. Noisy machines: Understanding noisy neural networks and enhancing robustness to analog hardware errors using distillation. *arXiv preprint arXiv:2001.04974*, 2020.

[14] Pulakesh Upadhyaya, Xiaojing Yu, Jacob Mink, Jeffrey Cordero, Palash Parmar, and Anxiao Andrew Jiang. Error correction for hardware-implemented deep neural networks. 2019.

[15] Kunping Huang, P. Siegel, and Anxiao Jiang. Functional error correction for robust neural networks. *IEEE Journal on Selected Areas in Information Theory*, 1:267–276, 2020.

[16] C. Huang, Y. Li, and L. Dolecek. Acoco: Adaptive coding for approximate computing on faulty memories. *IEEE Transactions on Communications*, 63(12):4615–4628, 2015.

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

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.