CodeNet: Training Large Scale Neural Networks in Presence of Soft-Errors

Sanghamitra Dutta [1] **Ziqian Bai** [2] **Tze Meng Low** [1] **Pulkit Grover** [1]

Abstract

We propose a novel strategy to make distributed training of DNNs resilient to computing errors, a problem that has remained unsolved despite being first posed in 1956 by von Neumann. He also speculated that the efficiency and reliability of the human brain are achieved by allowing for low-power but error-prone components with redundancy for error-resilience. It is surprising that this problem remains open, even as massive artificial neural networks are being trained on increasingly low-cost and unreliable processing units. Our coding-theory-inspired strategy, CodeNet, addresses this problem by providing a unified strategy for error-resilient DNN training without significant overhead. The overheads of coding are kept low by obviating the need to re-encode the updated parameter matrices after each iteration from scratch. Furthermore, CodeNet is completely decentralized with no central node (single point of failure), allowing all primary computational steps to be error-prone. We provide the first theoretical result that demonstrates that when accounting for checkpointing, CodeNet significantly reduces the expected computation time over replication. Our experiments show that CodeNet achieves the best accuracy-runtime tradeoff compared to both replication and uncoded strategies. CodeNet is a significant step towards biologically plausible neural network training, that could hold the key to orders of magnitude efficiency improvements.

1. Introduction

Inspired by the success of Shannon's theory of information (Shannon, 1948) in addressing errors in communication, and the remarkable efficiency and speed of the human brain in processing information with seemingly error-prone components, von Neumann began the study of computing in presence of noisy computational elements in 1956 (von Neumann, 1956), as is also evident from the influence of the McCulloch-Pitts model of a neuron (McCulloch & Pitts, 1943) in his work. It is often speculated that the error-prone nature of brain's hardware helps it be more efficient: rather than paying the cost at a component level, it may be more efficient to accept component-level errors, and utilize sophisticated error-correction mechanisms for overall reliability of the computation (Barlow, 1961; Yang et al., 2017). The brain operates at a surprisingly low power of about 15 W (Yanushkevich et al., 2013), and attains high accuracy and speeds, despite individual neurons in the brain being slow and error-prone (Sreenivasan & Fiete, 2011; Olshausen & Field, 1996; Barlow, 1961). Even today, the brain's system-level energy requirement is orders of magnitude smaller than the most efficient computers, despite substantial efforts in imitating the brain, going as far as using spiking neural networks (Merolla et al., 2014). While there is growing interest in training using low-cost and unreliable hardware (Courbariaux et al., 2015; Sakr et al., 2019; Sala et al., 2017), they still use significantly more power-consuming and reliable components than that used in the brain. Thus, von Neumann's original motivation of training neural networks in presence of noise and errors still remains open today. Towards addressing this important intellectual question, this work provides a unified strategy for error-resilience in every operation during the training of Deep Neural Networks (DNNs) *without significant overhead.*

Neural Networks, proposed in mid 1900s (Rosenblatt, 1958; Rumelhart et al., 1986), have revolutionized modern machine learning and data mining. However, training large-scale neural networks with millions of parameters (Krizhevsky et al., 2012; He et al., 2016) often requires large training time exceeding a few days. The ever-increasing size of neural networks creates a pressing demand for resources and power for fast and reliable training. In our experiments, that appear later, we demonstrate that ignoring errors entirely during DNN training can severely degrade the performance[1], even with a probability of error as low as 3×10^{-4} (see Fig. 2). Instead, by embracing errors in computing, one may be able to reduce the power budget of each individual computational node. These low-power

[1]Carnegie Mellon University [2]Simon Fraser University. Correspondence to: S Dutta <sanghamd@andrew.cmu.edu>.

[1]Perturbations (in small magnitudes) can sometimes be useful in non-convex minimization. But, soft-errors can cause bit-flips in most-significant bits, and the error-magnitudes can be quite large.

error-prone components enabled with an overall system-level-error-correction might hold the key to orders of magnitude improvements in efficiency while providing fast and reliable training as long as the overhead is kept small.

Related Works: Fault tolerance has been actively studied since von Neumann's work (see (Pippenger et al., 1991; Spielman, 1996; Yang et al., 2017)). One popular technique is *checkpointing* (Herault & Robert, 2015), where the computation-state is stored in a disk at regular intervals and the last stored state is retrieved when errors are detected. However, this has immense communication costs that can slow down the computation if the errors are too frequent.

An alternative (and often complementary) approach is forward error correction where redundancy is introduced into the computation itself, and detected errors are corrected prior to proceeding. There is no need to roll back if the number of errors are limited. Use of sophisticated (*i.e.*, non-replication) error-correcting codes in forward error correction dates at least as far back as 1984, when Algorithm-Based-Fault-Tolerance (ABFT) techniques (Huang & Abraham, 1984; Herault & Robert, 2015) were proposed for certain linear algebraic operations. Recently, "Coded Computation" (Cadambe & Grover, 2017; Lee et al., 2016; Yu et al., 2017; Aktas et al., 2017; Tandon et al., 2017; Lee et al., 2017; Dutta et al., 2018b; Karakus et al., 2017; Reisizadeh et al., 2017; Charles & Papailiopoulos, 2018; Wang et al., 2015) has emerged as an evolution on ABFT to address the problem of stragglers[2] using erasure codes.

Key Novelties: In this paper, we propose *CodeNet*, a novel strategy that enables fast and reliable training of Deep Neural Networks (DNNs) in distributed and parallelized architectures that use unreliable processing components. We advance on ideas from information and coding theory to design novel error-correction mechanisms that use redundancy to compute reliably in presence of "soft-errors," *i.e.*, undetected errors that can corrupt the computation of a node, producing garbage outputs that are far from the true (noiseless) output (Li et al., 2007). The main significance of CodeNet lies in ensuring that the additional overheads due to coding are kept low and comparable to replication, since matrices are not required to be encoded afresh even though they update at each iteration. Only vectors are encoded at each iteration which is much cheaper computationally.

CodeNet is completely decentralized, allowing for all primary computational operations to be error-prone, including the nonlinear step which is an obstacle because most techniques of coding in computing are linear. Even the error-detection and decoding are allowed to be erroneous and are accomplished in a decentralized manner, by replicating the functionality at multiple nodes and introducing some very low-complexity verification steps that are assumed to be error-free. Error-prone detection and decoding is important because of two reasons: (a) it avoids having a single point-of-failure in the system (if central node fails, so does the algorithm) and allows for all computations to be error-prone including encoding/decoding/nonlinear activation/Hadamard product; (b) it is also an important requirement for *biological plausibility* of any neural computation algorithm (see e.g. (Olshausen & Field, 1996)).

[2]Stragglers refer to a few slow nodes that can delay the entire computation as the master has to wait for all the nodes to finish.

System Model: A DNN consists of L weight matrices $\{\boldsymbol{W}_l\}_{l=1 \text{ to } L}$, one for each layer, that are trained and updated at each iteration based on a single data point and its label using stochastic gradient descent. For ease of explanation, assume that they are square matrices of dimensions $N \times N$. For completeness, we briefly include the primary computational steps of training here:
Feedforward stage: For l=1 to L,
[O1] $\boldsymbol{s}^l = \boldsymbol{W}^l \boldsymbol{x}^l$ (Matrix-vector product).
[C1] $\boldsymbol{x}^{(l+1)} = f(\boldsymbol{s}^l)$ (Nonlinear Activation).
Backpropagation stage: For l=L to l=1,
[O2] $(\boldsymbol{c}^l)^T = (\boldsymbol{\delta}^l)^T \boldsymbol{W}^l$ (Matrix-vector product).
[C2] $(\boldsymbol{\delta}^{(l-1)})^T = (\boldsymbol{c}^l)^T \circ g(\boldsymbol{x}^l)$ (Hadamard product).
Update stage: [O3] $\boldsymbol{W}^l \leftarrow \boldsymbol{W}^l + \eta \boldsymbol{\delta}^l (\boldsymbol{x}^l)^T$.

We are provided with P *base* nodes, along with few redundant nodes such that: every node can store only a $\frac{1}{P}$ fraction of *each* of the L weight matrices. Thus, the storage per node for layer l is $\frac{N^2}{P}$. There is provision for error-free checkpointing after every I_0 iterations, but the time taken for checkpointing (τ_{cpt}) or retrieving (τ_f) is much larger compared to the runtime of an error-free iteration of training (τ_b). Errors can happen at any node, at any stage in the computation. When an error occurs during any step, the node produces a garbage output, *i.e.*, the true output corrupted by additive Gaussian noise. Any additional computation introduced, e.g., encoding, decoding also become prone to errors themselves. However, the longer a computation, the more error-prone it is assumed to be (Li et al., 2007).

Our primary goal is to design a unified, model-parallel DNN training strategy resilient to errors in steps O1, O2 and O3 (complexity $\Theta(N^2)$), while keeping the additional communication complexities as well computational overheads significantly low. It is desirable that the occurrence of errors is always detected even if they are too many to be corrected. If the errors are within the error-tolerance, then they should be corrected; otherwise the strategy should be able to declare a "decoding failure" and revert to the last checkpoint. As a secondary goal, it is desirable to incorporate error-resilience in all other error-prone steps (complexity $\ll \Theta(N^2)$) and make the strategy decentralized. Here, we address the primary goal. Please see the expanded version (Dutta et al., 2019) for proofs and also discussion on the secondary goal.

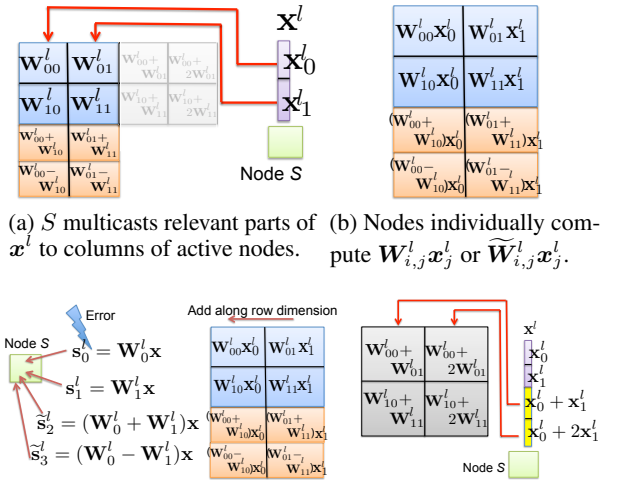
2. Main Results

Theorem 1 (Error Tolerance)**.** *Under the probabilistic error model, CodeNet*(m,n) *uses a total of* $\hat{P}=P+n(t+1)+m(t+1)$ *nodes to detect and correct any* t *errors after both steps O1 and O2, at a single layer during an iteration with probability* 1*. Moreover, if there are more errors, it is able to declare a decoding failure with probability* 1 *even if it cannot correct them.*

For intuition, we briefly discuss some key steps of CodeNet. Before the start of training, the matrix $\boldsymbol{W}^l$ is divided into $m\times n$ blocks (denoted by $\boldsymbol{W}^l_{i,j}$). These blocks are encoded vertically and horizontally using an $(m+t+1,m)$ and an $(n+t+1,n)$ systematic MDS code respectively (see the layout in Fig. 1a). Each node stores one block, that is either systematic ($\boldsymbol{W}^l_{i,j}$) or parity ($\widetilde{\boldsymbol{W}}^l_{i,j}$). This matrix encoding is performed only once. In all subsequent iterations, as we discuss afterwards, the coded sub-matrices (or blocks) are able to update themselves by encoding vectors (at low-complexity) instead of matrices. For now, assume that every node has the updated sub-matrix $\boldsymbol{W}^l_{i,j}$ or $\widetilde{\boldsymbol{W}}^l_{i,j}$ for that iteration to start with (Asm 1). For ease of explanation, also assume there is an error-free virtual master node S that has $\boldsymbol{x}^l$ before the feedforward stage at layer l (Asm 2).

In the feedforward stage, only the nodes corresponding to the row-wise MDS code are active to perform the matrix-vector product $\boldsymbol{W}^l\boldsymbol{x}^l$ (see Fig. 1). Under the probabilistic error model, an $(m+t+1,m)$ MDS code can correct t errors with probability 1, and if there are more errors, it is able to declare a decoding failure with probability 1 (proved in Theorem 3 (Dutta et al., 2018a)). Therefore, after step O1, the virtual master S aggregates the outputs from all the $m+t+1$ rows and performs error-detection (parity check of complexity $\Theta(tN)$). It begins to perform decoding only if it detects some errors, and is able to successfully decode $\boldsymbol{s}^l(=\boldsymbol{W}^l\boldsymbol{x}^l)$ if the errors are within t. Otherwise it declares a decoding failure and reverts back to the last checkpoint. When the errors are within t, the master applies the function $f(\cdot)$ on $\boldsymbol{s}$ (complexity $\Theta(N)$) to generate $\boldsymbol{x}^{l+1}$ for the next layer (validating our assumption Asm 2) and repeats the steps of Fig. 1 for the next layer.

A similar technique is followed for the backpropagation stage. A critical observation here is that in the update stage, all worker nodes now already have the required sub-vectors that they need to update themselves. For example, the node having $\boldsymbol{W}^l_{00}$ gets $\boldsymbol{x}^l_0$ in the feedforward stage (see Fig. 1a) and $\boldsymbol{\delta}^l_0$ in the backpropagation stage (elaborated in (Dutta et al., 2019)), and is able to update itself as $\boldsymbol{W}^l_{00}\leftarrow\boldsymbol{W}^l_{00}+\eta\boldsymbol{\delta}^l_0(\boldsymbol{x}^l_0)^T$. For the parity nodes, this is enabled by a low-complexity additional encoding step that *only involves encoding vectors* (see Fig. 1d). This also validates our assumption Asm 1 that every node is able to update themselves without encoding matrices afresh at each iteration, and ensures that *the additional overheads are low* (see (Dutta et al., 2019) for a formal result). *Errors in step O3 are also detected/corrected after step O1 or O2 of the next iteration when the affected node first produces an output.* CodeNet can thus detect *and correct* errors with a lower resource overhead as compared to two-way-replication which requires $\hat{P}=2P$ nodes to only detect errors. We leverage this to demonstrate speedup in expected time next.



(a) S multicasts relevant parts of $\boldsymbol{x}^l$ to columns of active nodes. (b) Nodes individually compute $\boldsymbol{W}^l_{i,j}\boldsymbol{x}^l_j$ or $\widetilde{\boldsymbol{W}}^l_{i,j}\boldsymbol{x}^l_j$.

(c) Active nodes compute the sum of partial results and send to S. (d) S multicasts coded sub-vectors to inactive nodes.

Figure 1. Feedforward Stage in CodeNet (m=n=2, t=1).

Theorem 2. *The ratio of the expected time taken to complete* M *iterations by replication to CodeNet scales as*

$$\frac{\min_{I_0}\frac{M}{I_0}\tau_{cpt}+\frac{M}{I_0}(\tau_f p_0+\tau_b(1-p_0))\frac{\frac{1}{(p_0)^{I_0}}-1}{\frac{1}{(p_0)}-1}}{\min_{I_0}\frac{M}{I_0}\tau_{cpt}+\frac{M}{I_0}(\tau_f p_0+\tau_b(1-p_0))\frac{\frac{1}{(p_0+p_1)^{I_0}}-1}{\frac{1}{(p_0+p_1)}-1}},$$

where p_0 *is the probability of no error and* p_1 *is the probability of the set of error patterns correctable by CodeNet.* The advantage arises because replication proceeds forward only with probability p_0 while CodeNet proceeds forward with probability p_0+p_1. CodeNet requires much less frequent checkpointing and retrieval as compared to replication. We conclude with some experiments in Fig. 2.

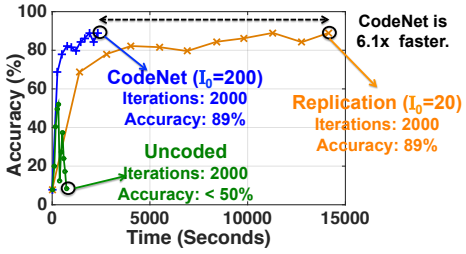


Figure 2. We train a 3 layer DNN [784 10^4 10^4 10] on MNIST using Amazon EC2. Each node has an independent error probability 0.0003, during O1, O2 or O3. CodeNet uses 38 nodes. A comparable replication strategy, with equal storage per node, uses more nodes (40) but takes longer to achieve the same accuracy. An uncoded strategy with no error-resilience has poor accuracy. CodeNet therefore has the best accuracy-runtime trade-off.

References

Aktas, M., Peng, P., and Soljanin, E. Effective Straggler Mitigation: Which Clones Should Attack and When? *ACM SIGMETRICS Performance Evaluation Review*, 45 (2):12–14, 2017.

Barlow, H. B. Possible principles underlying the transformations of sensory messages. *Sensory Communication*, pp. 217–234, 1961.

Cadambe, V. and Grover, P. Codes for Distributed Computing: A Tutorial. *IEEE Information Theory Society Newsletter*, 67(4):3–15, December 2017.

Charles, Z. and Papailiopoulos, D. Gradient coding via the stochastic block model. *arXiv preprint arXiv:1805.10378*, 2018.

Courbariaux, M., Bengio, Y., and David, J.-P. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 3123–3131, 2015.

Dutta, S., Bai, Z., Jeong, H., Low, T. M., and Grover, P. A unified coded deep neural network training strategy based on generalized polydot codes for matrix multiplication. *arXiv preprint arXiv:1811.10751*, 2018a.

Dutta, S., Fahim, M., Haddadpour, F., Jeong, H., Cadambe, V., and Grover, P. On the optimal recovery threshold of coded matrix multiplication. *arXiv preprint arXiv:1801.10292*, 2018b.

Dutta, S., Bai, Z., Low, T. M., and Grover, P. Codenet: Training large scale neural networks in presence of soft-errors. *arXiv preprint arXiv:1903.01042*, 2019.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Herault, T. and Robert, Y. *Fault-Tolerance Techniques for High Performance Computing*. Springer, 2015.

Huang, K. H. and Abraham, J. A. Algorithm-based fault tolerance for matrix operations. *IEEE Transactions on Computers*, 100(6):518–528, 1984.

Karakus, C., Sun, Y., Diggavi, S., and Yin, W. Straggler Mitigation in Distributed Optimization through Data Encoding. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 5440–5448, 2017.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 1097–1105, 2012.

Lee, K., Lam, M., Pedarsani, R., Papailiopoulos, D., and Ramchandran, K. Speeding up distributed machine learning using codes. In *IEEE International Symposium on Information Theory (ISIT)*, pp. 1143–1147, 2016.

Lee, K., Suh, C., and Ramchandran, K. High-dimensional coded matrix multiplication. In *IEEE International Symposium on Information Theory (ISIT)*, pp. 2418–2422, 2017.

Li, X., Shen, K., Huang, M. C., and Chu, L. A Memory Soft Error Measurement on Production Systems. In *USENIX Annual Technical Conference*, pp. 275–280, 2007.

McCulloch, W. S. and Pitts, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

Merolla, P. A., Arthur, J. V., Alvarez-Icaza, R., Cassidy, A. S., Sawada, J., Akopyan, F., Jackson, B. L., Imam, N., Guo, C., Nakamura, Y., Brezzo, B., Vo, I., Esser, S. K., Appuswamy, R., Taba, B., Amir, A., Flickner, M. D., Risk, W. P., Manohar, R., and Modha, D. S. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014. ISSN 0036-8075. doi: 10.1126/science.1254642. URL http://science.sciencemag.org/content/345/6197/668.

Olshausen, B. A. and Field, D. J. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607, 1996.

Pippenger, N., Stamoulis, G., and Tsitsiklis, J. On a lower bound for the redundancy of reliable networks with noisy gates. *IEEE Transactions on Information Theory*, 37(3): 639–643, May 1991. ISSN 0018-9448. doi: 10.1109/18.79921.

Reisizadeh, A., Prakash, S., Pedarsani, R., and Avestimehr, A. S. Coded computation over heterogeneous clusters. In *IEEE International Symposium on Information Theory (ISIT)*, pp. 2408–2412, 2017.

Rosenblatt, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning representations by back-propagating errors. *Nature*, 323(6088):533, 1986.

Sakr, C., Wang, N., Chen, C.-Y., Choi, J., Agrawal, A., Shanbhag, N., and Gopalakrishnan, K. Accumulation bit-width scaling for ultra-low precision training of deep networks. *arXiv preprint arXiv:1901.06588*, 2019.

Sala, F., Kabir, S., Van den Broeck, G., and Dolecek, L. Don't fear the bit flips: Optimized coding strategies for binary classification. *arXiv pre-print arXiv:1703.02641*, 2017.

Shannon, C. A mathematical theory of communication. *Bell Syst. Tech. J.*, 27(4):623–656, Oct 1948. ISSN 0005-8580. doi: 10.1002/j.1538-7305.1948.tb00917.x.

Spielman, D. A. Highly fault-tolerant parallel computation. In *Symposium on Foundations of Computer Science*, pp. 154–163, Oct 1996.

Sreenivasan, S. and Fiete, I. Error correcting analog codes in the brain: beyond classical population coding for exponentially precise computation. *Nature Neuroscience*, 14: 1330–1337, 2011.

Tandon, R., Lei, Q., Dimakis, A. G., and Karampatziakis, N. Gradient Coding: Avoiding Stragglers in Distributed Learning. In *International Conference on Machine Learning (ICML)*, pp. 3368–3376, 2017.

von Neumann, J. Probabilistic logics and the synthesis of reliable organisms from unreliable components. *Automata studies*, 34:43–98, 1956.

Wang, D., Joshi, G., and Wornell, G. Using Straggler Replication to Reduce Latency in Large-scale Parallel Computing. *ACM SIGMETRICS Performance Evaluation Review*, 43(3):7–11, 2015.

Yang, Y., Grover, P., and Kar, S. Computing Linear Transformations With Unreliable Components. *IEEE Transactions on Information Theory*, 63(6):3729–3756, 2017.

Yanushkevich, S. N., Kasai, S., Tangim, G., and Tran, A. *Introduction to Noise-Resilient Computing*. Morgan & Claypool Publishers, 2013.

Yu, Q., Maddah-Ali, M. A., and Avestimehr, A. S. Polynomial Codes: an Optimal Design for High-Dimensional Coded Matrix Multiplication. In *Advances In Neural Information Processing Systems (NIPS)*, pp. 4403–4413, 2017.