REDUCING COMMUNICATION OVERHEAD IN DISTRIBUTED LEARNING BY AN ORDER OF MAGNITUDE (ALMOST)

Anders Øland and Bhiksha Raj

School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213
{anderso,bhiksha}@cs.cmu.edu

ABSTRACT

Large-scale distributed learning plays an ever-more increasing role in modern computing. However, whether using a compute cluster with thousands of nodes, or a single multi-GPU machine, the most significant bottleneck is that of *communication*. In this work, we explore the effects of applying quantization and encoding to the parameters of distributed models. We show that, for a neural network, this *can* be done – without slowing down the convergence, or hurting the generalization of the model. In fact, in our experiments we were able to reduce the communication overhead by nearly an order of magnitude – while actually improving the generalization accuracy.
Keywords: Neural Networks, Distributed Training, Compression

1. INTRODUCTION

With the sizes of today's datasets growing to giga- and petabytes, we see an increasing need for distributed solutions to learning problems. In particular, in the deep learning literature, we have seen the number of parameters in a single model reach the billions [1, 2]. This renders training on a single node infeasible - both with respect to memory requirements and execution time. It becomes necessary, and indeed imperative, to distribute the computation across many machines.

However, distributed computing requires communication: data and model parameters must be exchanged between the worker machines that collaborate on the computation. Even using the fastest interconnects, such as InfiniBand (IB) or PCI Express 3.0 (PCIe), this communication overhead can become a serious bottleneck and greatly slow down the learning.

In this paper we address the problem of reducing communication overhead in distributed learning of large neural networks.

Communication overhead arises both from the need to transfer *data* to the workers, and the need to share model parameters among them. In our work we focus on the latter, minimizing the communication overhead due to exchange of model parameters. Specifically, we focus on gradient-based methods in a *data-parallel* setting [3]; *i.e.* where the dataset is distributed across workers, and each worker has a complete copy of the model. During training, each worker locally updates its own model, and the updated model parameters must periodically be synchronized among all of the workers.

The general approach to minimizing communication overhead has been just to reduce the *number* of parameters that are exchanged, *e.g.* by having fewer parameters in the first place by sparsifying them, exchanging only some of the parameters, or by sending them less frequently [4, 2].

We take a different approach – rather than impoverishing the number of parameters exchanged, we reduce the communication overhead by reducing the *number of bits* used to transmit each parameter. We achieve this by quantizing the transmitted values. The entropy [5] of the parameters of the network vary with training epoch, and the layer of the network that they represent. To take advantage of this, we vary the number of bits used to quantize any value dynamically, based on the observed entropy of the parameter values in any layer, at any epoch. Additional compression is obtained through Huffman coding [6] of the parameters prior to transmission. The additional overhead of conveying the information about the quantization levels and Huffman code dictionaries, so that the machines receiving these communicated values can decode them, is insignificant compared to the actual number of bits needed to represent the parameters. We are thus able to achieve a compression of nearly an order of magnitude in the communication overhead, for no loss of generalization error in the trained network, as evaluated on a standard classification task.

In the process, we also achieve a second, surprising result. The introduction of quantization noise appears to have a beneficial effect on the training. When the parameters are quantized to a slightly higher bit rate than that required to maintain generalization error, the resulting network actually achieves *lower* generalization error than that obtained with unquantized transmission of parameters. Moreover, the training often appears to converge faster. In effect, we simultaneously achieve reduced communication overhead, improved generalization error, and faster convergence by quantizing the parameters for communication.

The proposed method can also be coupled with techniques that only transmit a subset of the parameters. Here too the parameters could be further compressed as we propose, resulting in large reductions in the communication overhead.

The rest of the paper is structured as follows. In Section 2 we briefly outline the training setup. In Section 3 we discuss the rationale behind our compression scheme. In Section 4 we outline our compression algorithm. In Section 5 we provide experimental analysis of our solution, and in Section 6 we conclude with discussion.

2. DISTRIBUTED TRAINING SETUP

Our experiments were performed on a multi-GPU architecture: five NVIDIA GTX 780s connected by a single PCIe 3.0 bus. We used a custom neural network (NN) implementation written in a combination of Cuda and MATLAB; GPU-GPU copying was enabled by a single MEX file that wraps a call to *cudaMemCpyPeer*. A *worker* in this scenario, is thus a dedicated CPU thread controlling one GPU.

The NN was trained using standard Backprop with SGD and mini-batches, exponential learning rate decay, and light ℓ_1 regularization. As mentioned, we have focused solely on the *data-parallel* setting, where each worker has a complete copy of the model, but works on an exclusive segment of the dataset. For sake of consistency and repeatability of our experiments, we chose a *synchronous*

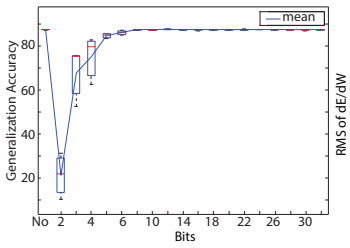


Fig. 1. Generalization accuracy vs. quantization level of weights.

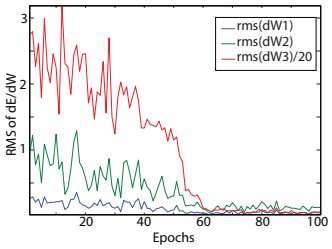


Fig. 2. RMS MSE derivative w.r.t. weights vs. training epoch.

weight exchange scheme. That is, after each epoch the workers exchange weights, such that they all end up having the exact same *averaged* weights. This exchange can be done in various ways, depending on the context, and it can be done *asynchronously* too [1]. Our proposed method is however agnostic to all of this, as each transmission of parameters, from worker to worker (or e.g. slave to master node), is completely self-contained. When a worker transmits something, it must send along some quantization and encoding meta-data; the receiver then uses that meta-data to decode and dequantize the signal. Whenever a worker changes anything, e.g. averages the weights it received with its own, and sends along the results - it must first adapt the quantization and encoding meta-data; i.e. the *codebook* etc. is not static.

3. PARAMETER COMPRESSION

Our solution to reducing communication overhead is to *compress* parameters before transmission. In choosing a compression scheme, we must consider many factors including **cost** (complexity and execution time), **memory usage**, **effectiveness** (compression ratio), and **impact on learning** (the generalization error of the model). Obviously, the cost of encoding, sending, and decoding the data must be less than simply sending the data as is. Similarly, cutting down the cost of communication is meaningless if it affects the learning negatively, such that either the generalization error increases, or it takes considerably more iterations to converge (so that nothing is gained with respect to the total execution time).

The model parameters to be transmitted are single- or double-precision floating point numbers. We find simple lossless compression of these parameters through algorithms such as LZW [7] to generally be ineffective. They achieve little, if any, reduction in the size of the data, at a significant computational cost.

Instead, we will utilize the robustness of model parameters to minor perturbation, particularly during training, to compose an inexpensive *lossy* compression scheme through quantization. Secondly, we will utilize the relatively narrow range within which most parameter values lie to further compress the parameters through a lossless compression scheme.

In the following subsections all illustrations are based on experiments on the MNist database [8] with a deep neural network comprising 784 units in the input layer and three subsequent layers of size 392, 50 and 10 neurons respectively. The network is fully connected between any two consecutive layers. There are thus three sets of weights of size 784×392, 392×50 and 50×10 respectively. The activation functions of all neurons were tanh non-linearities.

3.1. Quantizing the parameters

As a first try, we attempted to simply quantize all parameters to a fixed number of bits. To quantize the values to N bits, we find the largest and minimum values of all weights in each epoch, and evenly split the range up into 2^N bins. Each weight is quantized to the center of the bin that it falls into. In terms of communication, it will require only N bits to transmit any weight.

Figure 1 shows results obtained with quantization to various levels. The plot shows the classification accuracy obtained with the fully trained network, as a function of the number of bits used to quantize the weights. The boxes show the mean, median (red line) and standard deviation of the results obtained from 1500 runs of training with different initializations. The accuracy obtained with 8-bit quantization is comparable to that obtained without quantization. This compares very favorably with the 32 bits typically required to represent floating point numbers in IEEE format.

3.2. Dynamic Selection of Quantization Levels

As network training progresses, the derivative of the mean squared error (MSE) being minimized with respect to the weights converges towards a small value approaching zero, albeit in a somewhat noisy manner. Figure 2 shows the root-mean squared (RMS) value of the derivatives of the error with respect to network weights for each of the three layers in our network as a function of epoch. The derivatives for the third layer are scaled by 20 to fit in the plot. We note that the derivatives converge towards zero for all layers. Also, the derivatives are generally larger at higher layers of the network.

The derivatives are directly indicative of the degree of quantization that can be tolerated by the weights. When the derivatives are large, relatively small perturbations of weights can result in relatively large changes in the error. On the other hand, when the derivatives are small, the network is tolerant to larger perturbations of the weights. We also know that quantization to a larger number of bits results in smaller expected quantization error, while quantization to a fewer number of bits results in larger quantization error.

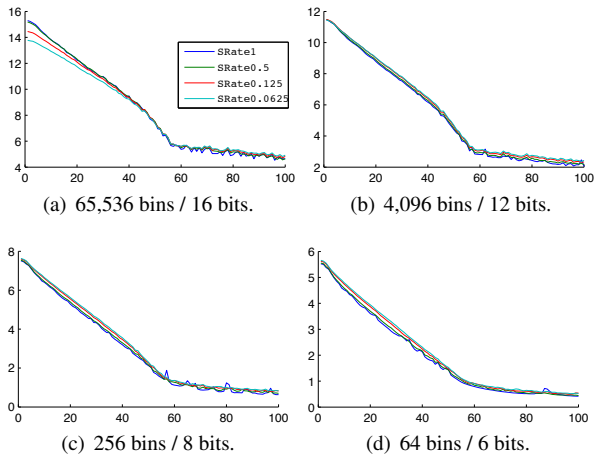
These observations together suggest that when the RMS value of the derivative is large, a larger number of bits are required to quantize the weights. On the other hand, at small RMS values of the derivative, a smaller number of quantization levels will suffice.

This leads us to propose a *dynamic* selection of the number of bits to quantize the weights. In each epoch, for each layer, we will choose the number of bits to quantize the weights in that layer according to the RMS value of the derivative of the MSE with respect to the weights. Specifically, assuming the tolerance to changes in the MSE to be some constant T, and the RMS value of the derivatives to be G, the bin size ΔW that may be expected to perturb the objective to within T is given by $\Delta W \propto T/(G + c_0)$, where c_0 is a floor that enforces an upper bound on ΔW. Consequently, the number of bins that the weights are quantized to, $N_{bin} \propto \Delta W^{-1} \Rightarrow N_{bin} \propto (G+c_0)$. The number of bits required to quantize the weights comes out to $N = \log N_{bin} = \log(G + c_0) + c_1$. c_1 may be viewed as a floor on the number of bits used to quantize the weights. To ensure that N does not go below this value, c_0 must be set to 1.0.

3.3. Lossless Compression

The distribution of the weights is not uniform in any layer. Figure 3 shows the estimated entropy of the network weights in the various layer of the network, as a function of epoch. In each case, for N-bit quantization we have computed the entropy from the normalized histogram of the weights over the 2^N quantization bins. The four panels shows the entropy obtained with different levels of quantization of the weights. Expectedly, increasing the number of bits used to quantize the weights increases their entropy; however the overall trend of the entropy remains the same in all cases.

The entropy is generally less than N, the number of bits used to quantize the weights, and decreases with the epochs. This indicates



(a) 65,536 bins / 16 bits. (b) 4,096 bins / 12 bits.

(c) 256 bins / 8 bits. (d) 64 bins / 6 bits.

Fig. 3. Computing the entropy using different sample ratios and number of histogram bins (bits). The sample ratio (denoted SRate in the legend), representing the fraction of the total set of weights that was used to estimate the entropy, has very little affect. Lowering the number of bins merely results in a shift of the entire curve.

that lossless Huffman coding of the quantized weights can result in significant compression of the weights.

3.4. Bits Required to Quantize the Weights

Comparing Figure 3 and Figure 2 we note that the entropy is well correlated with the RMS value of the gradient. Empirically, we find the normalized correlation between $\log(G + c_0)$ for $c_0 = 1$ and the entropy of the weights to be greater than 0.75 at all times, and frequently higher than 0.95, particularly in the higher layers.

This correlation is sufficient for us to use the entropy of the weights as a proxy for the RMS value in determining the desired quantization. Thus we arrive at the following estimate for the number of bits required to quantize the weights in the k^{th} level of the network in the e^{th} training epoch: $N_{k,e} = H_{k,e} + c$, where $H_{k,e}$ is the entropy of the weights in the k^{th} layer, in epoch e, and c is a floor on the number of bits to be used.

The above formula appears self-referential at first glance: the entropy estimate $H_{k,e}$ itself depends on the number of bits used to quantize the weights as noted earlier. To resolve this, we compute $H_{k,e}$ from a preliminary quantization of the weights to M bits. Typically, it is sufficient to estimate $H_{k,e}$ from only a small sample of the complete set of weights. Empirically, we have found that using as few as 3% of the total set of weights can provide us with reliable estimates of the entropy of the weights in any layer. M is a parameter we can choose. c must be empirically determined, and now also accounts for the fact that $H_{k,e}$ depends on M, since increasing M by a factor of 2^K will increase $H_{k,e}$ by approximately K bits. On the other hand, decreasing M results in smoother, and potentially more robust estimates of the entropy as seen from Figure 3.

4. AN ALGORITHM FOR DYNAMIC COMPRESSION OF NETWORK WEIGHTS

Algorithm 1 describes our final algorithm for compresing the network weights. The output of the algorithm is a set of packaged weights, where the package contains all the necessary information required to decode the quantized weights to their actual values. Algorithm 2 describes the algorithm used to decode the packaged weights to real numbers that can be used by the receiving node. *HuffmanCodebook*, *Encode* and *Decode* are standard algorithms for computing the Huffman codebook, encoding a sequence of bits with a given Huffman code, and decoding a stream of Huffman codes.

The parameters F, M, and c must be empirically determined. F is the fraction of weights that are used to obtain the preliminary estimate of entropy, H. M is the size in bits of the initial quantizer used to obtain this preliminary estimate. c is the quantization floor. Empirically, we have that $F = 0.03$, $M = 4$, and $c = 6$ provide excellent results. We present experiments that support this conclusion in the next section.

Algorithm 1 Weight Compression Algorithm

Input: Weights $\mathcal{W}_l$, $l = 1..L$ for each of the L layers in the net, F, M, c.
for layer $l = 1..L$ **do**
  Find $w^l_{max} = \max_{w \in \mathcal{W}_l} w$ and $w^l_{min} = \min_{w \in \mathcal{W}_l} w$.
  Entropy H = **Estimate Entropy**$(\mathcal{W}, F, w_{min}, w_{max}, M)$
  Compute the number of bits: $N = H + c$.
  Quantize $\mathcal{W}_l$: $\mathcal{Q}_l$ = **Quantize**$(\mathcal{W}_l, w^l_{max}, w^l_{min}, N)$.
  Estimate Probability $\mathbf{P}_l$ = **Estimate Distribution**$(\mathcal{Q}_l, N)$
  Huffman Codebook $\mathcal{C}_l$ = *HuffmanCodebook*$(\mathbf{P}_l)$.
  Encode $\mathcal{Q}_l$: $\mathcal{E}_l = Encode(\mathcal{Q}_l, \mathcal{C}_l)$.
  Package $\mathcal{P}_l = [\mathcal{E}_l, w^l_{min}, w^l_{max}, \mathbf{P}_l, N]$.
end for
Output: $\mathcal{P}_l$, $l = 1..L$.

function QUANTIZE$(\mathcal{W}, w_{min}, w_{max}, K)$
  Return $\mathcal{Q} = \{ \left\lfloor \frac{2^N (w_{max} - w)}{w_{max} - w_{min}} \right\rfloor \; \forall \, w \in \mathcal{W} \}$
end function

function ESTIMATE DISTRIBUTION$(\mathcal{Q}, K)$
  Compute a 2^K-bin histogram $h(i), i = 1..2^K$ from $\mathcal{Q}$.
  Normalize the histogram : $p(i) = \frac{h(i)}{\sum_i h(i)}$.
  Return $\mathbf{P} = \{p(i),\ i = 1..2^K\}$
end function

function ESTIMATE ENTROPY$(\mathcal{W}, F, w_{min}, w_{max}, M)$
  Select a random subset $\mathcal{W}_{sel}$ of $\mathcal{W}$ of size $F|\mathcal{W}|$.
  Quantize $\mathcal{W}_{sel}$: $\mathcal{Q}_{sel}$ = **Quantize**$(\mathcal{W}_{sel}, w_{min}, w_{max}, M))$
  Estimate $\mathbf{P}$ = **Estimate Distribution**$(\mathcal{Q}_{sel}, M)$
  Return $H = -\sum_{i=1}^{2^M} p(i) \log p(i)$.
end function

Algorithm 2 Weight De-Compression Algorithm

Input: $\mathcal{E}_l, w^l_{min}, w^l_{max}, \mathbf{P}_l, N$
Huffman Codebook $\mathcal{C}_l$ = *HuffmanCodebook*$(\mathbf{P}_l)$.
Decode $\mathcal{Q}_l = Decode(\mathcal{E}_l, \mathcal{C}_l)$.
$\mathcal{W}_l = \{w_{min} + \frac{(w_{max} - w_{min})(i + 0.5)}{2^N} \; \forall \, I \in \mathcal{Q}_l\}$.
Output: $\mathcal{W}_l$

5. EXPERIMENTS

We ran experiments to investigate the proposed method and demonstrate its validity. The experimental setup was that described in Section 2. Experiments were run on the 784-392-50-10 neural network described in Section 3, on the MNIST database. In total, we ran the Backpropagation algorithm about 1,500 times, while saving all the parameters of the NN after each of the 100 epochs.

5.1. Establishing Optimal Parameter Values

The proposed algorithm has three parameters: (a) the sample ratio F, which determines the fraction of the weights we use to arrive at H, the preliminary estimate of entropy, (b) the quantization floor c, and (c) M, the bit size used for the preliminary entropy estimate. We investigate the setting of each of these.

Estimating F: Figure 3 shows the entropy estimates obtained with different sample rates in each panel. The entropy estimates do not vary much with sample rate. We therefore set $F = 0.03$, *i.e.* 3%. For larger networks an even lower value of F may suffice.

Estimating M and c: The optimal values of M and c are closely coupled. First we establish the *ceiling* on the number of bits needed, through fixed-size quantization. Figure 1 shows the generalization accuracies obtained with fixed-size quantization at every quantization level between 2 and 30 bits. We are able to recover baseline results at 8 bits, which gives us a 75% reduction in communication (in single-precision) by quantization alone. We therefore consider a ceiling of 10 bits to provide room for variation.

Ideally, we must explore the entire range of (M, c) values to establish the optimal setting. Instead, we present a summary that only considers marginal variation of each of the variables, while assuring the reader that the conclusions noted generalize to the larger search.

In general, we can expect the dynamically assigned quantization size not to exceed the bit-rate ceiling. The maximum value of the preliminary entropy estimate obtained from M-bit quantization is M. Thus, we expect $c + M$ to be no greater than the ceiling. We use this to first establish an effective value for M. Figure 4a shows the generalization performance obtained on the test set when $c + M = 10$. Each point on the plot represents the aggregate statistics of 50 trials run with different initializations. We have varied the floor c from 2 to 10 bits. A ceiling of 10 and a floor of 10 (represented by "10:10" in the figure) represents fixed quantization to 10 bits. We observe that we obtain the best results with $M = 4$, representing a quantization floor of $c = 6$ bits. Figure 4b shows the performance obtained at different values of the floor c, at $M = 4$. At $M = 4, c = 5$ we obtain performance just marginally below baseline, giving us a "sweet spot" in terms of compression. It is interesting to note that the performance with 9:5, representing $M = 4$ is actually superior to that obtained with 10:5 representing $M = 5$.

Thus, we establish $c = 5$, $M = 4$, since this results in only minor loss of performance. Figure 5 shows the average bit-rate per parameter as a function of epoch. This represents an average bit rate of 6.8 bits/parameter over all training epochs, and an overall compression rate of 4.70.

Finally, we apply the final stage of Huffman coding to the quantized weights. Figure 5 also shows the average bit-rate obtained with Huffman coding. This results in a significant reduction of bit-rate at every stage, and an overall average bit-rate of 3.55 bits/parameter, representing an overall compression rate of 9.01 with respect to the baseline with no compression.

5.2. Going beyond the Baseline

Figure 4 reveals an interesting fact. The generalization error obtained with $c = 6, M = 4$ is actually *superior* to the baseline. These results are consistent over a large number of runs of the experiment. The mean bit rate at this setting is 3.78 bits/parameter, representing an overall compression of 8.47.

Figures 4c and 4d show the number of iterations required for the training to achieve convergence, where we define converegence as the peaking of generalization accuracy. In all cases compression of the weights results in faster convergence. At $c = 6, M = 4$, convergence is achieved in 20% fewer iterations than the baseline. Considering both the gains from parameter compression and the faster convergence, we obtain an effective overall reduction of communication of a factor of 10.32, while also improving generalization error.

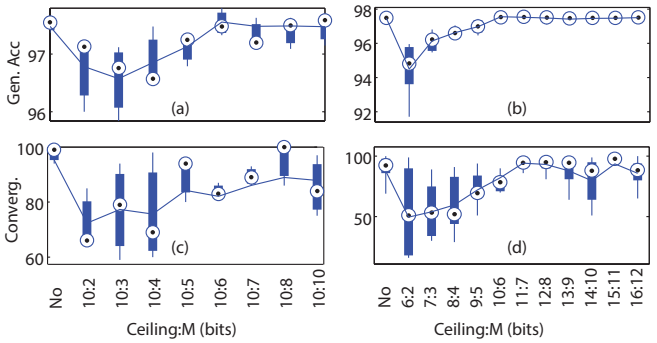


Fig. 4. **Left:** Generalization accuracies (a), and convergence (c) for fixed quantization ceiling of 10 bits with varying floor. **Right:** Generalization accuracies (b), and convergence (d) for a fixed quantization *range* of 4 bits with increasing floor. **All:** The left-most value is the baseline, and the dots are medians and the line plots the mean.

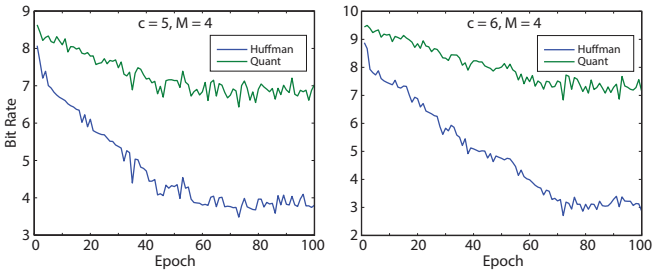


Fig. 5. Bit-rate as a function of epoch with raw and Huffman coded quantized values. **Left:** at $c = 5, M = 4$. **Right:** at $c = 6, M = 4$.

6. CONCLUSION & DISCUSSION

As the size of neural networks and the data used to train them increase, not only will more parameters need to be communicated, they will be done so over increasingly greater numbers of machines. Gains such as those we report will become very important. The computational overhead of compression is miniscule compared to the actual transmission in these scenarios. Although we have presented the compression scheme in the context of neural networks, it will apply to gradient-descent based distributed optimimzation in general. Moreover, it can also be combined with approaches that only transmit a subset of parameters, for additive gains in compression.

While our results are very promising, the actual numbers reported must still be considered with caution. A part of our current work is validating these results on much larger corpora. We are also investingating extensions to *model-parallel* formalisms [2].

The improved generalization results we observe are, perhaps, not so surprising, since the effect of quantization is to introduce quantization noise. It is well-known that noise may help a learning algorithm escape from local minima [9], and has been used in de-noising Auto-Encoders (DAEs) [10], and other variants such as dropout [11] and DropConnect [12]. However, it remains to confirm our results on larger data sets and networks; this is work in progress. We do not expect any surprises.

7. REFERENCES

[1] Q. V. Le, Ranzato M., Monga R., Devin M., Chen K., Corrado G.S., and et. al., "Building high-lvel features using large scale unsupervised learning," Tech. Rep., arqiv.org, December 2011.

[2] Adam Coates, Brody Huval, Tao Wang, David Wu, Bryan Catanzaro, and Ng Andrew, "Deep learning wit h COTS HPC systems," pp. 1337–1345, 2013.

[3] D. W. Hillis and G. L. Steele, "Data parallel algorithms," *Communications of the ACM*, vol. 29, no. 12, pp. 1170–1183, 1986.

[4] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, and Andrew Y Ng, "Large Scale Distributed Deep Networks," *Advances in neural . . .* , pp. 1223–1231, 2012.

[5] C E Shannon and W Warren, *The Mathematical Theory of Communication*, Univ. of Ill. Press, 1949.

[6] D A Huffman, "A Method for the Construction of Minimum-Redundancy Codes," in *Proceedings of the IRE*. 1952, pp. 1098–1101, IEEE.

[7] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *IEEE Transaction on Information Theory*, vol. 24, no. 5, pp. 530–536, 1978.

[8] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[9] Tom M Heskes and Bert Kappen, "On-line learning processes in artificial neural networks," in *North-Holland Mathematical Library*, pp. 199–233. Elsevier, 1993.

[10] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol, "Extracting and composing robust features with denoising autoencoders," in *the 25th international conference*, New York, New York, USA, 2008, pp. 1096–1103, ACM Press.

[11] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv.org*, July 2012.

[12] Li Wan, Matthew Zeiler, Sixin Zhang, Yann L Cun, and Rob Fergus, "Regularization of neural networks using dropconnect," in *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, 2013, pp. 1058–1066.