Training Neural Networks in Parallel With ADMM

Nicholas Haltmeyer
hanicho1@umbc.edu

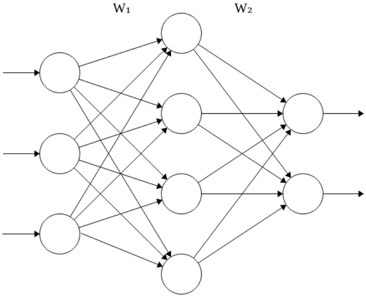
Department of Computer Science and Electrical Engineering
University of Maryland, Baltimore County

August 25, 2023

Background

What are neural networks?

- Connectionist model used in machine learning
- Used for function approximation, classification, control, and others
- Maps inputs to outputs using concurrent units



Notation

A network consists of L layers, with each layer having a *weight* matrix, W_l, and a non-linear *activation function*, h_l, where $l = 1, 2, \ldots, L$.
Given a_{l-1}, a layer produces $a_l = h_l(W_l a_{l-1})$ as output.
Final output given by

$$f(a_0; W) = W_L h_{L-1}(\ldots W_2 h_1(W_1 a_0)). \tag{1}$$

Learning problem is to minimize some cost function, C, for weights:

$$\underset{W}{\text{minimize}} \quad C(f(a_0; W), y). \tag{2}$$

Motivation

- Traditional methods (e.g., backpropagation) do not scale well
  - Relies on many cheap iterations to converge, making update synchronization expensive
  - Gradient-based methods suffer from the vanishing gradient problem, where information is lost before getting to the early layers
- Solution: use alternating direction updates (Taylor et al., 2016) to operate on subsets of data for each process
  - Yields significant performance boost when the amount of training data is large
  - Performs well with deep networks (large values of l)

Alternating Direction Method of Multipliers

Constrained optimization method that works by splitting variables and imposing an equality constraint (Lions and Mercier, 1979).

$$\underset{x}{\text{minimize}} \quad f(x) + g(x), \tag{3}$$

becomes

$$\begin{aligned} \underset{x,y}{\text{minimize}} \quad & f(x) + g(y) \\ \text{subject to} \quad & x = y. \end{aligned} \tag{4}$$

Solve for x with y constant, then y with x constant, to convergence.

ADMM for Neural Networks

Split the variables in the original minimization problem by introducing $z_l = W_l a_{l-1}$, $a_l = h_l(z_l)$.

$$\begin{aligned} \underset{\{W_l\},\{a_l\},\{z_l\}}{\text{minimize}} \quad & C(z_L, y) \\ \text{subject to} \quad & z_l = W_l a_{l-1}, \qquad \text{for } l = 1, 2, \ldots, L \\ & a_l = h_l(z_l), \qquad \text{for } l = 1, 2, \ldots, L-1. \end{aligned} \tag{5}$$

Relax the constraints using ℓ^2 terms and Lagrangian:

$$\begin{aligned} \underset{\{W_l\},\{a_l\},\{z_l\}}{\text{minimize}} \quad & C(z_L, y) + \langle z_L, \lambda \rangle + \beta_L ||z_L - W_L a_{L-1}||^2 + \\ & \sum_{l=1}^{L-1} \left[\gamma_l ||a_l - h_l(z_l)||^2 + \beta_l ||z_l - W_l a_{l-1}||^2 \right], \end{aligned} \tag{6}$$

Weight Updates

Weights are updated by

$$W_l \leftarrow \arg\min_{W} ||z_l - W a_{l-1}||^2, \tag{7}$$

with a solution in

$$W_l \leftarrow z_l a_{l-1}^{\dagger}. \tag{8}$$

- Note that $a_{l-1}^{\dagger} = a_{l-1}^{T}(a_{l-1}a_{l-1}^{T})^{-1}$

Activation Updates

Activations are updated by

$$a_l \leftarrow \arg\min_a \beta_l||z_{l+1} - W_{l+1}a||^2 + \gamma_l||a - h_l(z_l)||^2, \tag{9}$$

with a solution in

$$a_l \leftarrow (\beta_{l+1}W_{l+1}^T W_{l+1} + \gamma_l I)^{-1}(\beta_{l+1}W_{l+1}^T z_{l+1} + \gamma_l h_l(z_l)). \tag{10}$$

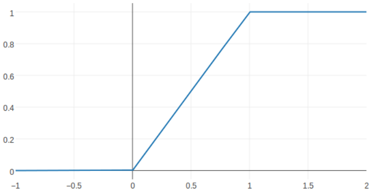
Output updates

For $l = 1, 2, \ldots, L-1$, outputs take

$$z_l \leftarrow \arg\min_z \gamma_l ||a_l - h_l(z)||^2 + \beta_l ||z - W_l a_{l-1}||^2. \tag{11}$$

Activation function is assumed to be entry-wise over components of z. A good choice of $h_l(z)$ with a closed form solution is a discrete sigmoid function:

$$h(x) = \begin{cases} 1 & x \geq 1, \\ x & 0 < x < 1, \\ 0 & x \leq 0. \end{cases}$$



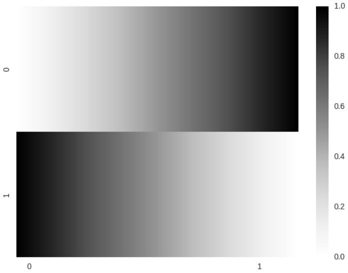
Output updates (cont.)

For $l = L$, outputs take

$$z_L \leftarrow \arg\min_z C(z, y) + \langle z, \lambda \rangle + \beta_L ||z - W_L a_{L-1}||^2. \tag{12}$$

Similar to the activation function, cost function must be chosen carefully. Hinge cost is used here:

$$C(x, y) = \begin{cases} \max(0, 1 - x) & y = 1, \\ \max(0, x) & y = 0. \end{cases}$$



Lagrange Updates

Lagrange updates given by

$$\lambda \leftarrow \lambda + \beta_L(z_L - W_L a_{L-1}). \tag{13}$$

In practice, a small number of iterations are taken without updating λ, so that the initially random weights can settle to relative stability before setting λ.

Final Algorithm

Algorithm 1 ADMM for NNs

procedure ADMM-LEARN(features a_0, labels y)
  repeat
    for $l = 1, \cdots, L-1$ **do**
      $W_l \leftarrow z_l a_{l-1}^{\dagger}$
      $a_l \leftarrow (\beta_l W_{l+1}^T W_{l+1} + \gamma_l I)^{-1}(\beta_l W_{l+1}^T z_{l+1} + \gamma_l h_l(z_l))$
      $z_l \leftarrow \arg\min_z \gamma_l ||a_l - h_l(z)||^2 + \beta_l ||z - W_l a_{l-1}||^2$
    $W_L \leftarrow z_L a_{L-1}^{\dagger}$
    $z_L \leftarrow \arg\min_z C(z,y) + \langle z, \lambda \rangle + \beta_L ||z - W_L a_{L-1}||^2$
    $\lambda \leftarrow \lambda + \beta_L (z_L - W_L a_{L-1})$
  until converged

Parallelization

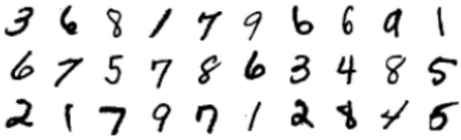
- Training data are scattered to N nodes
- $\{a_l\}$, $\{z_l\}$, λ are computed independently on each node
- Weights are updated using transpose reduction (Goldstein et al., 2015)
    - Compute subterms on every node
    - Take sum with MPI_Allreduce
    - Compute matrix inverse, followed by product

$$W_l \leftarrow \left(\sum_{n=1}^{N} z_l^n (a_{l-1}^n)^T \right) \left(\sum_{n=1}^{N} a_{l-1}^n (a_{l-1}^n)^T \right)^{-1} \tag{14}$$

Experiments

Evaluated using the MNIST (Lecun et al., 1998) database of handwritten digits.

- 28×28 grayscale pixel images
- Pixel values from 0 to 255, normalized to be from 0 to 1
- Digit labels from 0 to 9
- 60,000 training samples, 10,000 validation samples
- Error is ratio of incorrectly categorized validation samples



Settings

- 3-layer network used in evaluation ($784 \times 500 \times 500 \times 10$)
- Programmed in C, using the GNU Scientific Library and MPI[1]
- Run on Intel MPI version 5.1.3 build 20160120, compiled with icc version 16.0.3 build 20160415
- -O3 compiler flag to perform optimization
- Using the maya compute cluster with 72 nodes, each with 64GB memory and two 8 core Intel E5-2650v2 Ivy Bridge CPUs (2.6 GHz, 20 MB cache)

[1] `https://github.com/hanicho/admm-nn`

Initialization

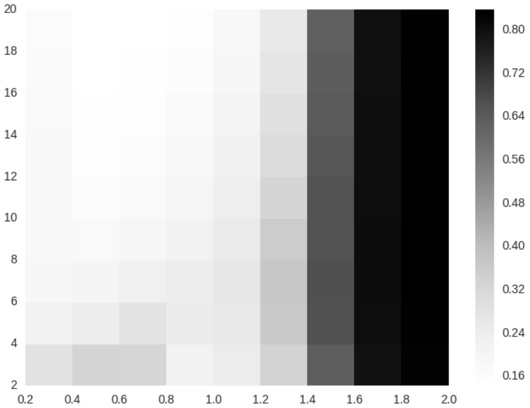
Weights are initialized through a normalized initialization scheme: (Glorot and Bengio, 2010)

$$W_l \sim U[-\frac{\sqrt{6}}{\sqrt{n_l + m_l}}, \frac{\sqrt{6}}{\sqrt{n_l + m_l}}]. \tag{15}$$

Where n_l, m_l are the number of input and output units for l. This starts weights close to zero, preventing any one from being dominant.
10 iterations are taken before first updating λ.

Hyperparameters

- $\{\beta_l\}$, $\{\gamma_l\}$ chosen to be constant for all layers
- $\beta = 0.2$, $\gamma = 20$ found to work well



Memory Prediction

Each node has a complete copy of the weight matrix, and a local subset of $\{a_l\}$, $\{z_l\}$, λ.

- Weights: $8 \times \left(784 \times 500 + 500 \times 500 + 500 \times 10\right) = 5,176,000$ bytes
- Activations:
$$8 \times \left(\frac{60,000}{N} \times 500 + \frac{60,000}{N} \times 500 + \frac{60,000}{N} \times 10\right) = \frac{484,800,000}{N} \tag{16}$$
bytes
- Outputs: Same as activations
- Total: $5.176 + \frac{974.4}{N}$ MB

Memory Observation

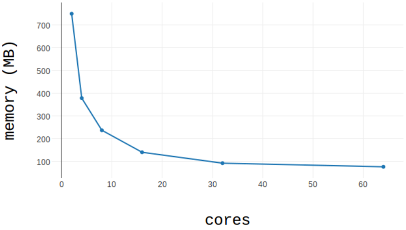
Table: Memory usage per node (MB).

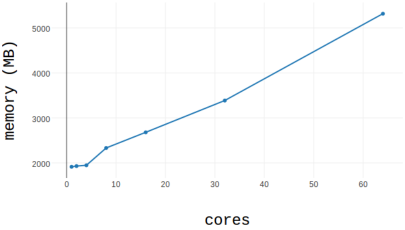
N	Predicted	Observed
1	979.58	1914.98
2	492.38	749.27
4	248.78	378.30
8	126.98	236.91
16	66.08	140.17
32	35.63	92.18
64	20.40	76.35

Table: Memory usage for all nodes (MB).

N	Predicted	Observed
1	979.58	1914.98
2	984.75	1930.69
4	995.10	1948.09
8	1015.81	2332.35
16	1057.22	2679.99
32	1140.03	3386.55
64	1305.66	5315.53

Memory Observation (cont.)



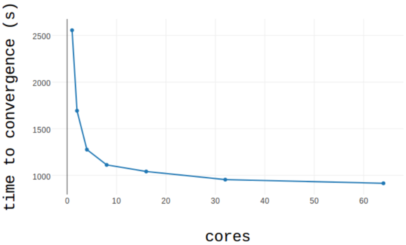


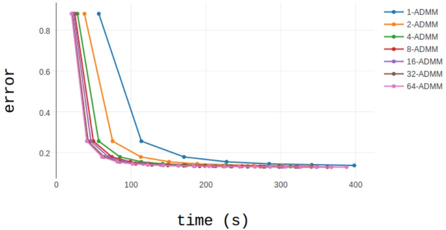
Parallel Performance

Table: Time to convergence (s)

N	Time	Speedup
1	2556.77	1.00
2	1692.83	1.51
4	1275.98	2.00
8	1111.96	2.30
16	1041.19	2.46
32	953.96	2.68
64	914.73	2.80

Parallel Performance (cont.)





Performance Bottleneck

Given the size of the MNIST dataset, speedup is bottlenecked when computing

$$\left(\sum_{n=1}^{N} a_{l-1}^{n}(a_{l-1}^{n})^{T}\right)^{-1}. \tag{17}$$

This occurs when the number of inputs for each layer is not significantly less than the number of samples stored on each node, clamping the wall time of each iteration around the time it takes to compute (17).

Future Work

- More thorough analysis of layer-dependent values of $\{\beta_l\}$, $\{\gamma_l\}$ may improve performance.
- Other means of acceleration typical in gradient-based methods may be applicable to ADMM (momentum, regularization)
- Evaluation over larger datasets would yield a better understanding of large-scale parallelization

Conclusions

- Applied ADMM to neural networks
- Derived appropriate parameters for the MNIST application domain
- Evaluated memory use and parallel speedup
- It works!

Acknowledgments

This project was done in collaboration with my advisor Dr. Ting Zhu. The hardware used in the computational studies is part of the UMBC High Performance Computing Facility (HPCF). The facility is supported by the U.S. National Science Foundation through the MRI program (grant nos. CNS–0821258 and CNS–1228778) and the SCREMS program (grant no. DMS–0821311), with additional substantial support from the University of Maryland, Baltimore County (UMBC). See `hpcf.umbc.edu` for more information on HPCF and the projects using its resources.

References

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Aistats*, volume 9, pages 249–256.

Goldstein, T., Taylor, G., Barabin, K., and Sayre, K. (2015). Unwrapping ADMM: efficient distributed computing via transpose reduction. *CoRR*, abs/1504.02147.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324.

Lions, P.-L. and Mercier, B. (1979). Splitting algorithms for the sum of two nonlinear operators. *SIAM Journal on Numerical Analysis*, 16(6):964–979.

Taylor, G., Burmeister, R., Xu, Z., Singh, B., Patel, A., and Goldstein, T. (2016). Training neural networks without gradients: A scalable ADMM approach. *CoRR*, abs/1605.02026.