LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-PROC-819671

Heterogeneity, Hyperparameters, and GPUs: Towards Useful Transport Calculations Using Neural Networks

M. M. Pozulp, P. S. Brantley, T. S. Palmer, J. L. Vujic

February 19, 2021

The International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering (M&C 2021)
Raleigh, NC, United States
October 3, 2021 through October 7, 2021

Disclaimer

ANS M&C 2021 - The International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering · Raleigh, North Carolina · October 3–7, 2021

HETEROGENEITY, HYPERPARAMETERS, AND GPUS: TOWARDS USEFUL TRANSPORT CALCULATIONS USING NEURAL NETWORKS

Michael M. Pozulp[1,2], Patrick S. Brantley[1], Todd S. Palmer[3], and Jasmina L. Vujic[2]
[1]Lawrence Livermore National Laboratory,
[2]University of California, Berkeley, [3]Oregon State University

pozulp1@llnl.gov, brantley1@llnl.gov,
todd.palmer@oregonstate.edu, vujic@nuc.berkeley.edu

ABSTRACT

We use a neural network to solve the discrete ordinates neutron transport equation in slab geometry. We extend previous work which solved homogeneous medium problems by considering problems with spatial heterogeneity. We also include a hyperparameter study and a description of our GPU implementation along with the runtime we observed using Sierra GPUs at LLNL.

KEYWORDS: transport, neural networks, heterogeneity, hyperparameters, GPUs

1. INTRODUCTION

There has been widespread interest in neural networks (NNs) since at least 2012 when a NN known today as AlexNet made substantial progress in an image classification benchmark called ImageNet. Since then, NNs: surpassed human performance at Go, matched human performance at detecting some eye diseases and cancers, and became the leading technology for anomaly detection, recommender systems, and speech recognition. In all of these cases, NNs performed substantially better than any other algorithm and enabled the solution of problems that were previously unsolved.

NNs appear to be less researched for the solution of differential equations in the physical sciences. There are NN solutions to Burgers' equation, Schrödinger's equation, Laplace's equation, the slab geometry neutron diffusion equation [1], and the slab geometry discrete ordinates (S_N) neutron transport equation [2,3], but they only apply to specialized cases for which other solvers are easily used. So why consider NNs? A rapidly growing ecosystem of NN software and hardware exists which could be used to improve time to solution for transport problems. One example is the GPUs in LLNL's Sierra supercomputer [4], which are designed for fast, power-efficient execution of operations found in graphics rendering and neural network training.

At LLNL we typically use Monte Carlo (MC) or S_N for transport, *not* NNs. The goal is not to replace MC or S_N. Both will remain production capabilities for the foreseeable future at LLNL. Instead, we seek a foundation on which hybrid methods employing NNs in combination with MC or S_N may be considered in the future. The profitability of a hybrid method employing NNs

was recently demonstrated by one example in which a NN was used to accelerate a S_N solver by replacing a Gaussian Elimination step [5].

NNs were used to solve homogeneous medium neutron transport problems in [2,3]. We extend [2,3] by considering problems with spatial heterogeneity. We also include a hyperparameter study and a description of our GPU implementation along with the runtime we observed using a Sierra GPU at LLNL. Heterogeneity, a hyperparameter study, and GPU results are the new aspects of this work.

2. METHODS

We solve Eq. (1a), the slab geometry neutron transport equation, in a non-multiplying medium using a single energy group and linearly-anisotropic scattering with vacuum boundaries:

$$\mu\frac{\partial\psi(z,\mu)}{\partial z}+\Sigma_t(z)\psi(z,\mu)=\frac{1}{2}\left[\Sigma_{s0}(z)\phi_0(z)+3\Sigma_{s1}(z)\phi_1(z)\mu+Q(z)\right] \tag{1a}$$

$$\psi(z=0,\mu>0)=0 \tag{1b}$$

$$\psi(z=z_{\text{max}},\mu<0)=0. \tag{1c}$$

We updated the cross sections in the loss function from [3] to accommodate heterogeneity by adding spatial dependence, $[\boldsymbol{\Sigma_t},\boldsymbol{\Sigma_{s0}},\boldsymbol{\Sigma_{s1}}]\equiv[\Sigma_t(z),\Sigma_{s0}(z),\Sigma_{s1}(z)]$, to get this loss function:

$$\begin{aligned}\mathcal{L}=&\left\|\boldsymbol{\nabla}\hat{\boldsymbol{\Psi}}\text{diag}(\boldsymbol{\mu})+\boldsymbol{\Sigma_t}\hat{\boldsymbol{\Psi}}-\frac{1}{2}\left(\boldsymbol{\Sigma_{s0}}\hat{\boldsymbol{\Phi}}_0\mathbb{1}_N^T-3\boldsymbol{\Sigma_{s1}}\hat{\boldsymbol{\Phi}}_1\boldsymbol{\mu}^T-\boldsymbol{Q}\right)\right\|_F^2\\&+\gamma_L\|\hat{\boldsymbol{\Psi}}^{\mu>0}(z=0)-\boldsymbol{\Psi}_L\|_F^2\\&+\gamma_R\|\hat{\boldsymbol{\Psi}}^{\mu<0}(z=z_{\text{max}})-\boldsymbol{\Psi}_R\|_F^2,\end{aligned} \tag{2}$$

where $\hat{\boldsymbol{\Psi}}(\boldsymbol{z},\boldsymbol{\mu})\equiv\hat{\boldsymbol{\Psi}}\in\mathbb{R}^{d\times N}$ is a discretized approximation of the angular flux which has been sampled at d spatial points $\boldsymbol{z}\in\mathbb{R}^d$ and N quadrature points $\boldsymbol{\mu}\in\mathbb{R}^N$, $\text{diag}(\boldsymbol{\mu})\in\mathbb{R}^{N\times N}$ is a matrix with the ordered values of the ordinate angles μ_i on the diagonals, $\mathbb{1}_N^T$ is a row vector of N ones, and F denotes the Frobenius norm. The NN is used to estimate $\hat{\boldsymbol{\Psi}}$, allowing us to compute the scalar flux $\hat{\boldsymbol{\Phi}}_0$ and current $\hat{\boldsymbol{\Phi}}_1$ as

$$\hat{\boldsymbol{\Phi}}_0=\hat{\boldsymbol{\Psi}}\boldsymbol{w} \tag{3}$$

$$\hat{\boldsymbol{\Phi}}_1=\hat{\boldsymbol{\Psi}}\text{diag}(\boldsymbol{\mu})\boldsymbol{w}, \tag{4}$$

where $\boldsymbol{w}\in\mathbb{R}^N$ are the Gauss-Legendre quadrature weights. The loss function is simply the residual of the S_N equations plus terms that aim to improve the accuracy at the boundaries. We

use the Adam optimizer [6] in `PyTorch` [7] to minimize the loss function. Loss minimization is an iterative process which ends when we achieve our convergence criterion:

$$\left|\mathcal{L}^{(l+1)} - \mathcal{L}^{(l)}\right| < \epsilon, \tag{5}$$

where $\mathcal{L}^{(l)}$ and $\mathcal{L}^{(l+1)}$ are the losses computed at iteration l and $l+1$, respectively, and ϵ is some small constant, in our case 10^{-13}.

We use the network architecture from [3], shown in Fig. 1. The boxes are operations: Linear is a matrix vector multiplication plus entry-by-entry addition that uses `NumPy` broadcasting semantics to handle addition operands with different dimensions [8], and Tanh is a nonlinear activation function. z is the input vector of points along the spatial axis, w is a weight matrix and b is a bias vector. The number of parameters p is equal to the number of weights and biases, so this network has $5 + 5 + 20 + 4 = 34$ parameters. Let $N \equiv$ number of ordinates and $h \equiv$ number of hidden layer nodes, then $p(h, N) = (h + h) + (hN + N)$ where the two terms come from the Linear(1, h) and Linear(h, N) operations. In our default architecture, $h = 5$ and $N = 4$, which results in $p(5, 4) = 34$ parameters.

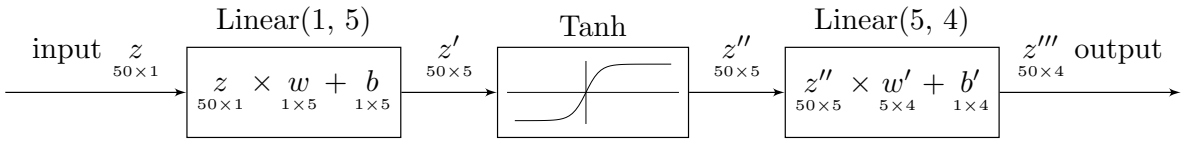


Figure 1: Default NN architecture.

The procedure for solving Eq. (1a) using the NN is summarized as follows:

1. Construct the NN and define the loss function to minimize.
2. Pass input data $\mathbf{z}$ through the network to yield the angular flux $\hat{\mathbf{\Psi}}$, the scalar flux $\hat{\mathbf{\Phi}}_0$, and the current $\hat{\mathbf{\Phi}}_1$.
3. Calculate the loss using Eq. (2).
4. Perform backpropagation by computing the gradient of the loss with respect to the input $\mathbf{z}$ (i.e., compute $\nabla_{\mathbf{z}}\mathcal{L}$) and use this to determine the gradient with respect to each parameter in the network.
5. Use the gradients computed above to update each parameter in the network such that the parameters are updated in the direction that decreases the loss the most.
6. Repeat the forward and backwards passes until the convergence criterion in Eq. (5) is achieved.

Our implementation is available on LLNL's Github [9].

3. RESULTS

3.1. SPATIAL HETEROGENEITY

This section contains numerical results from using the default NN architecture to solve six transport problems. Longer problem descriptions and problem parameters are included in APPENDIX A.

Problem 1. Full Slab Homogeneous purely-absorbing medium with a uniform source and vacuum boundaries.

Problem 2. Half Source Homogeneous absorbing and linearly-anisotropic scattering medium with a source on $[0, 0.5]$ and vacuum boundaries.

Problem 3. Reed Heterogeneous absorbing and scattering medium with multiple sources and vacuum boundaries. Four materials. Identical to Problem 1 from [10] except with a vacuum boundary at 0 instead of a reflecting boundary.

Problem 4. Sharp Slab 0 Heterogeneous strongly-absorbing medium with a source on $[0, 1]$ adjacent to a void on $[1, 5]$ and vacuum boundaries. Two materials.

Problem 5. Sharp Slab 1 Same as Problem 4 but with a weak absorber ($\Sigma_a = 1$) instead of a void. The ratio of the absorption cross sections of the two materials is 50.

Problem 6. Sharp Slab 5 Same as Problem 4 but with $\Sigma_a = 5$ and absorption cross section ratio 10.

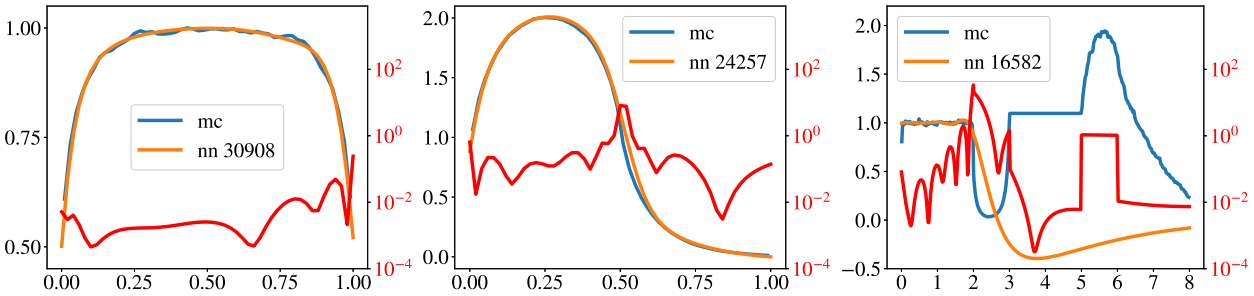
Fig. 2 shows our solutions. The horizontal axis is the spatial coordinate z and the left vertical axis is the scalar flux $\phi(z)$. The curves are labeled with the algorithm used to find the solution, where mc is Monte Carlo and nn is neural network. The integer in the nn label is the number of iterations required to achieve convergence. The red curve, which corresponds to the right vertical axis, is the spatial distribution of the loss, which has the same units as $\phi(z)$. The loss is calculated using Eq. (2).

In Fig. 2, the flux calculated using the NN very closely matches MC for **Problems 1-2** which are homogeneous medium problems. This is not the case for **Problems 3-6**. **Problems 4-6** demonstrate that the NN is not accurate for problems that include sharp gradients. The spatial gradient, which is a component of the neutron streaming term, is the first term in Eq. (1a). In the NN, we calculate this term using automatic differentiation (AD) in `PyTorch`. We were surprised by AD's sensitivity to sharp gradients and are working to understand it. We also want to try more traditional numerical methods for calculating the gradient, like finite differences.

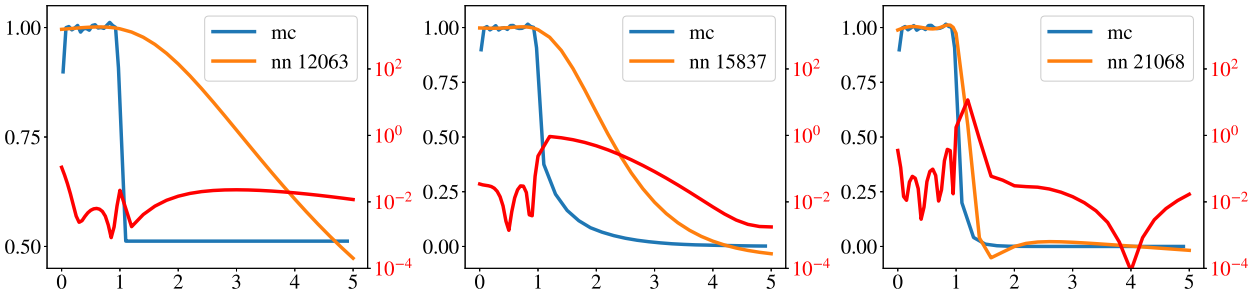
3.2. HYPERPARAMETER STUDY

This section contains numerical results from 25,469 solutions of **Problem 1**, where we changed some quality of the NN for each solution attempt.

In deep learning parlance, the weights and biases are the parameters of the NN, and any quantity that is not a weight or bias is a hyperparameter. Hyperparameters can affect convergence and iteration runtime, where convergence is defined as the number of iterations required to achieve the convergence criterion in Eq. (5). Table 1 lists the hyperparameters that we varied in this study, the



(a) Problem 1. Full Slab **(b) Problem 2. Half Source** **(c) Problem 3. Reed**



(d) Problem 4. Sharp Slab 0 **(e) Problem 5. Sharp Slab 1** **(f) Problem 6. Sharp Slab 5**

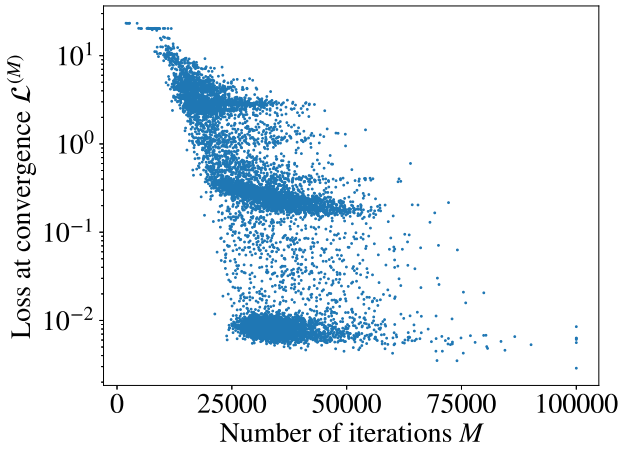
Figure 2: Scalar flux plots for six transport problems.

hyperparameters that we want to vary in a future study, and 3 hyperparameters which may not be applicable to our approach.

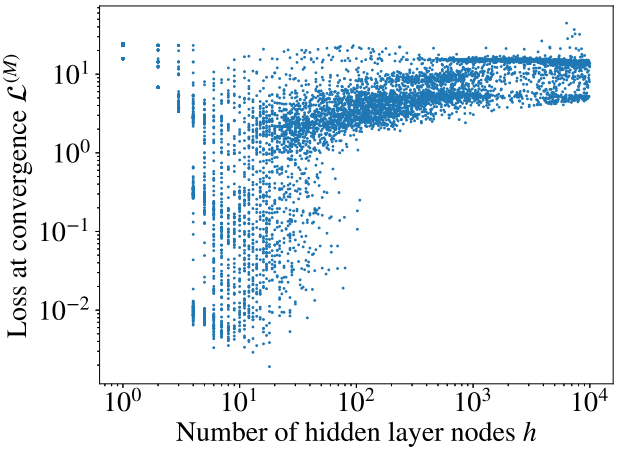
Table 1: Classification of hyperparameters.

Considered in this study	For future studies	Not applicable
RNG seed	Input vector size	Mini-batch size
Number of hidden layer nodes h	Number of hidden layers	Early stopping
Learning rate α	Optimizer-specific (eg β_1, β_2)	Dropout
Choice of optimizer	Regularization	
Choice of activation function	Initialization	
	Scaling inputs	

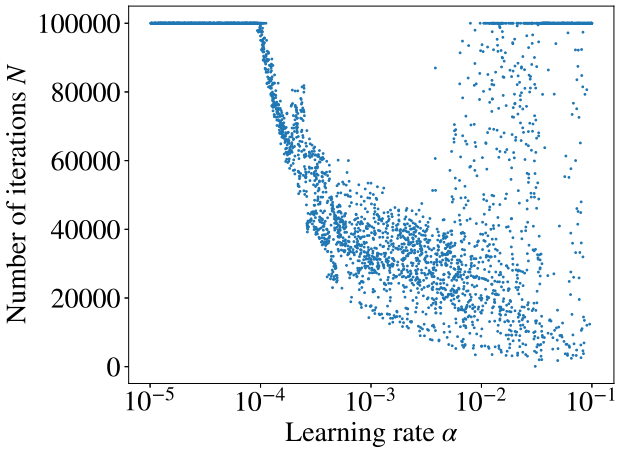
We sampled the hyperparameters with infinite domains from uniform distributions. The RNG seed samples were $\lfloor U(0, 2{<<}31) \rfloor$, the number of hidden layer nodes h samples were $\lfloor 10^{U(0,4)} \rfloor$, and the learning rate α samples were $10^{U(-5,-1)}$. The learning rate, also known as the step size, is a factor which determines the magnitude of the parameter adjustments made by an optimizer. Authors of optimizers publish recommended learning rates. For example, the Adam authors recommend 10^{-3}. Hyperparameters with finite domains were cycled through sequentially. For example, we sampled 9 optimizers $\approx \left\lfloor \frac{n}{9} \right\rfloor = \left\lfloor \frac{925}{9} \right\rfloor = 102$ times each, using a different RNG seed each time. Fig. 3 shows the effects of varying hyperparameters.



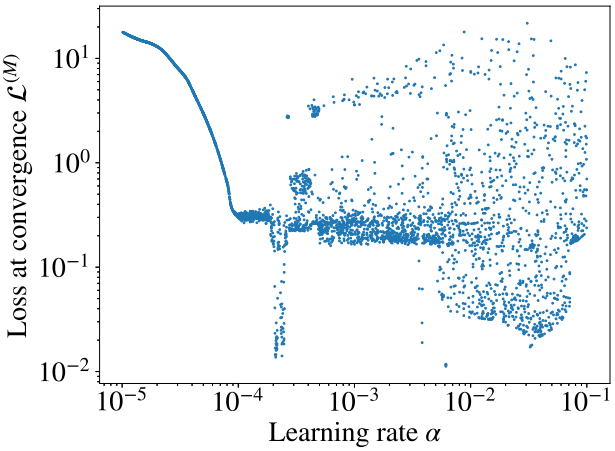
(a) Varied RNG seed only, $n = 10717$**.**



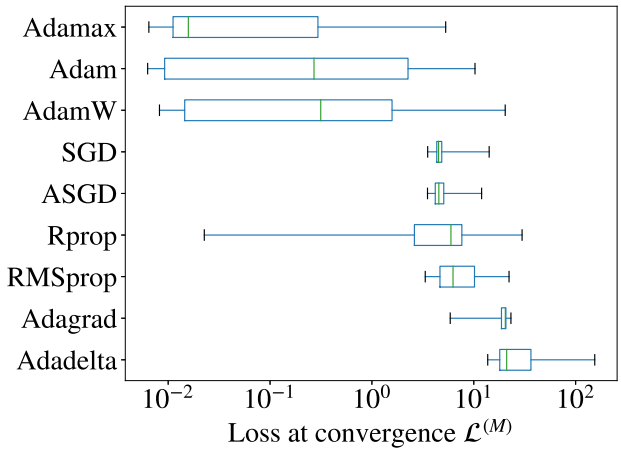
(b) Varied h **& RNG seed,** $n = 7666$**.**



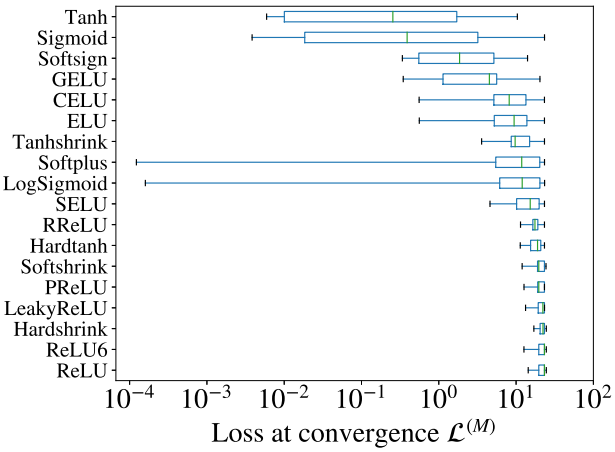
(c) Varied α **only,** $n = 3492$**.**



(d) Same data as plot (c).



(e) Varied optimizer & RNG seed, $n = 925$**.**



(f) Varied activation function & RNG seed, $n = 2669$**.**

Figure 3: Hyperparameter study results for Problem 1.

Row 1 of Fig. 3 shows that the loss at convergence $\mathcal{L}^{(M)}$ is sensitive to the RNG seed and the number of hidden layer nodes h. The RNG seed determines $\mathcal{L}^{(0)}$ because the NN parameters are initialized by sampling random variates from a uniform distribution, in our case $U(-1, 1)$. Fig. 3a shows that picking a different RNG seed can reduce $\mathcal{L}^{(M)}$ by three orders of magnitude. Fig. 3b shows that $\mathcal{L}^{(M)} < 10^{-2}$ requires $h \in [3, 13]$.

Row 2 of Fig. 3 shows $\mathcal{L}^{(M)}$ is sensitive to the learning rate. The maximum number of iterations that we allowed was 100,000, which is why Fig. 3c shows clumping at $M = M_{\text{max}} \equiv$ 100,000. Fig. 3c also shows that convergence may not be achievable before M_{max} for $\alpha < 10^{-4}$ and that $\alpha > 10^{-2}$ can hit M_{max}, but $\alpha \in (10^{-4}, 10^{-2})$ appears to always converge. Fig. 3d shows that the optimal α may be ≈ 0.0002 or $> 10^{-2}$ but this result is sensitive to the RNG seed. While not shown, we found that $\alpha \approx 10^{-3}$ is never worse than other α, and often optimal.

Row 3 of Fig. 3 shows $\mathcal{L}^{(M)}$ is sensitive to the choice of optimizer and the choice of activation function. Fig. 3e shows that Adam and its variants Adamax and AdamW can all achieve approximately the same minimum $\mathcal{L}^{(M)}$, but Adamax is more often closer to the minimum. Fig. 3f shows that Tanh and Sigmoid more often achieve a smaller $\mathcal{L}^{(M)}$ than other activation functions. Softplus and LogSigmoid achieved the smallest minimum $\mathcal{L}^{(M)}$, but the distance of their minimums from their medians may mean that Softplus and LogSigmoid only rarely provide such a great result.

It would be interesting to vary the NN architecture by simultaneously changing the input vector size, the number of nodes in the hidden layer, and the number of hidden layers to see how they affect convergence and iteration runtime. We think convergence could improve with a bigger NN, at the cost of increased iteration runtime. Another question is whether hyperparameter studies for other problems would lead to similar conclusions.

3.3. GPU EXECUTION

We used the `device` argument to `torch.tensor()` along with `torch.tensor.to()` to move tensors into GPU-accessible memory and `torch.nn.Sequential.cuda()` to move the model parameters and buffers into GPU-accessible memory. We ran on one node of LLNL's RZAnsel cluster [11], a small system with the same hardware as LLNL's Sierra supercomputer. The node has two IBM POWER9™ sockets each with 22 cores, and four NVIDIA Tesla™ V100-SXM2-16GB Volta™ GPUs. We ran **Problem 1** to convergence at 29599 iterations and 5 minutes and 44 seconds of walltime. The V100 flux closely matches the P9 flux, but the latter only took 2 minutes and 3 seconds to run, a slowdown of nearly 3x.

The work of a NN is serial with respect to the optimization iterations, so the available concurrency is limited by the number of parameters and the number of training data points. AlexNet had 60 million parameters and 1.2 million input images. The default architecture of our network has 34 parameters and only 1 input (see Fig. 1). The absence of significant parallel work was confirmed by `nvidia-smi`, which showed only 11% GPU utilization, and `nvprof` which showed that only 29.67 seconds were spent in GPU kernels whereas 153.11 seconds were spent in CUDA runtime API calls. The former is useful work and the latter is pure overhead. Table 2 and Table 3 show kernel and call times.

Table 2: Five most expensive GPU kernels per `nvprof`.

Time(%)	Time	Calls	Avg	Min	Max	Name
11.80	3.50177s	887996	3.9430us	3.8080us	5.1520us	_ZN2at6native6moder...
11.75	3.48571s	503200	6.9270us	4.9270us	10.272us	_ZN2at6native13redu...
8.42	2.49947s	680804	3.6710us	3.5830us	4.8640us	_ZN2at6native6moder...
7.24	2.14960s	266400	8.0690us	7.9360us	345.34us	volta_sgemm_32x32_s...
6.55	1.94456s	503200	3.8640us	3.7440us	5.0880us	_ZN2at6native6moder...

Table 3: Five most expensive CUDA runtime API calls per `nvprof`.

Time(%)	Time	Calls	Avg	Min	Max	Name
54.06	82.7777s	6127206	13.509us	10.640us	7.3316ms	cudaLaunchKernel
21.69	33.2049s	40848436	812ns	593ns	6.1128ms	cudaGetDevice
11.25	17.2308s	17997061	957ns	664ns	668.65us	cudaSetDevice
6.85	10.4897s	562472	18.649us	10.318us	49.983ms	cudaMemcpyAsync
1.90	2.91627s	7	416.61ms	10.307us	2.91571s	cudaMalloc

4. CONCLUSIONS

We presented the solution to six different slab geometry neutron transport problems using NNs. We also varied some hyperparameters and presented GPU results using a GPU from the Sierra supercomputer at LLNL. We were surprised by the challenge that sharp spatial gradients present to AD in `PyTorch`. We were also surprised by the sensitivity of $\mathcal{L}^{(M)}$ to $\mathcal{L}^{(0)}$. We understand that the 2x slowdown going from one P9 core to the V100 is due to a lack of parallel work. Resolving the gradient issue and GPU performance issue and expanding the hyperparameter study could be good directions for future work. If we add multigroup we could solve a two-material, two-group reactor problem in slab geometry.

ACKNOWLEDGEMENTS

The authors thank the LLNL WSC Research Coordination Council for funding this work through the LEARN program. The authors also thank the anonymous reviewers whose feedback led to a significantly improved final paper. This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

REFERENCES

[1] P. S. Brantley. "Artificial Neural Network Solutions of Slab-Geometry Neutron Diffusion Problems." *Trans Am Nuc Soc*, (83), p. 251 (2000).

[2] P. S. Brantley. "Spatial Treatment of the Slab-Geometry Discrete Ordinates Equations Using Artificial Neural Networks." In *Proceedings of M&C01*. Salt Lake City, UT, USA (2001).

[3] M. M. Pozulp. "1D Transport Using Neural Nets, SN, and MC." In *Proceedings of M&C19*, pp. 876–885. Portland, OR, USA (2019).

[4] S. S. Vazhkudai et al. "The Design, Deployment, and Evaluation of the CORAL Pre-exascale Systems." In *Proceedings of SC18*, volume 52, pp. 1–12. Dallas, TX, USA (2018).

[5] M. E. Tano and J. C. Ragusa. "Sweep-Net: An Artificial Neural Network for Radiation Transport Solves." *Journal of Computational Physics*, **volume 426** (2021).

[6] D. P. Kingma et al. "Adam: A Method for Stochastic Optimization." In *Proceedings of ICLR*. San Diego, CA, USA (2015).

[7] A. Paszke et al. "Automatic Differentiation in PyTorch." In *Proceedings of 31st NeurIPS*. Curran Associates, Inc., Long Beach, CA, USA (2017).

[8] R. C. Harris et al. "Array programming with NumPy." *Nature*, **volume 585**, pp. 357–362 (2020).

[9] "Narrows." (2020). URL https://github.com/llnl/narrows. LLNL-CODE-806068.

[10] W. H. Reed. "New Difference Schemes for the Neutron Transport Equation." *Nuclear Science and Engineering*, **volume 46**(2), pp. 309–314 (1971).

[11] "LLNL RZAnsel Cluster." (2020). URL https://hpc.llnl.gov/hardware/platforms/rzansel.

APPENDIX A. Problem Parameters

The following descriptions are to help the reader understand the problem setups for the flux calculations presented in Section 3.1. The problem inputs are available on LLNL's Github [9]. In particular, the file `test_mc21.py` runs the problems and generates the plots in Fig. 2.

Problem 1. Full Slab is a homogeneous and purely-absorbing slab with a source in the full domain and with vacuum boundaries. See Table 5 for parameters.

Problem 2. Half Source is a homogeneous slab with absorption and linearly-anisotropic scattering and a source on the left half of the domain and with vacuum boundaries. See Table 6 for parameters.

Problem 3. Reed is a four-material slab with a source in the first material and another source in the first centimeter of the fourth material and with vacuum boundaries. We used 200 zones, 50 in each material. The zones within a material are equal size, but the material extents differ so the size of the zones in one material is not the same as the size of the zones in another material. See Table 7 for parameters.

Problem 4, 5, 6. Sharp Slab 0, 1, 5 are two-material, purely-absorbing slabs with a source in the first material and vacuum boundaries. The first material is a strong absorber. The second material is a void ($\Sigma_t = 0$), a weak absorber ($\Sigma_t = 1$), or a moderate absorber ($\Sigma_t = 5$) for the problems named **Sharp Slab 0, 1, 5** respectively. The ratio of the absorption cross sections of the two materials is ∞, 50, 10 respectively. See Table 8 for parameters.

Table 4: Description of problem parameters.

Parameter	Description
Σ_t	Total cross section (cm^{-1})
Σ_{s0}	Total scattering cross section (cm^{-1})
Σ_{s1}	Linearly anisotropic cross section (cm^{-1})
Q	External source magnitude
J	Number of zones
N	Number of ordinates
P	Number of MC particles
h	Number of hidden layer nodes in NN

Table 5: Parameters used in Problem 1. Full Slab

	Σ_t	Σ_{s0}	Σ_{s1}	Q	J	N	P	h
$0 < z < 1$	8	0	0	8	50	4	1e6	5

Table 6: Parameters used in Problem 2. Half Source

	Σ_t	Σ_{s0}	Σ_{s1}	Q	J	N	P	h
$0 < z < 1$	8	6.4	1.6		50	4	1e6	5
$0 < z < 0.5$				8				
$0.5 < z < 1$				0				

Table 7: Parameters used in Problem 3. Reed

	Σ_t	Σ_{s0}	Σ_{s1}	Q	J	N	P	h
$0 < z < 8$			0			4	1e6	6
$0 < z < 2$	50	0		50	50			
$2 < z < 3$	5	0		0	50			
$3 < z < 5$	0	0		0	50			
$5 < z < 6$				1				
$5 < z < 8$	1	0.9		0	50			

Table 8: Parameters used in Problem 4, 5, 6. Sharp Slab 0, 1, 5

		Σ_t	Σ_{s0}	Σ_{s1}	Q	J	N	P	h
	$0 < z < 5$		0	0			4	1e6	5
	$0 < z < 1$	50			50	50			
Sharp Slab 0	$1 < z < 5$	0			0	50			
Sharp Slab 1	$1 < z < 5$	1			0	50			
Sharp Slab 5	$1 < z < 5$	5			0	50			