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

M&C 2021
Raleigh, North Carolina
October 3-7, 2021

Mike Pozulp
Patrick Brantley
Todd Palmer
Jasmina Vujic

October 5, 2021

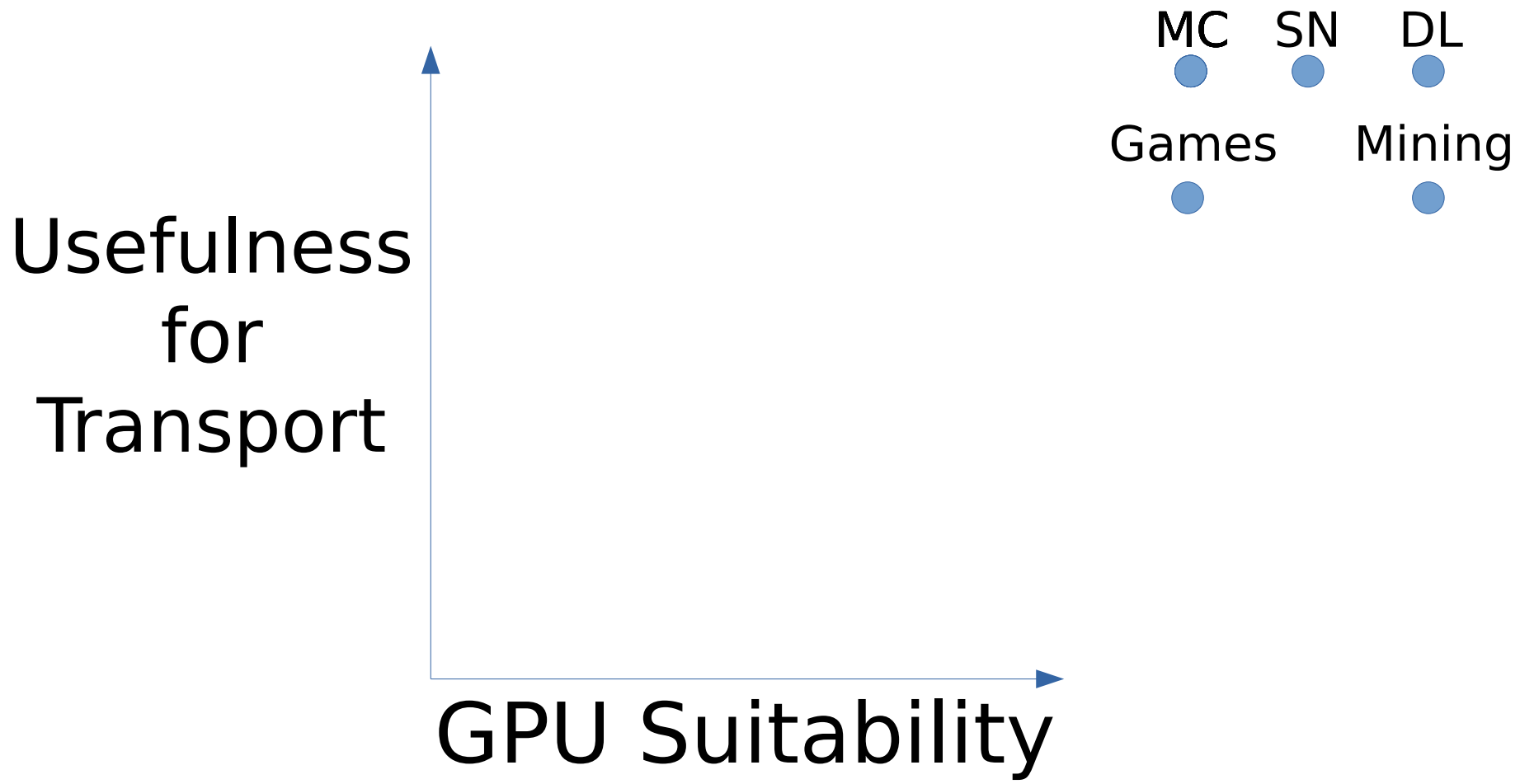
LLNL-PRES- 827462
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Lawrence Livermore National Laboratory

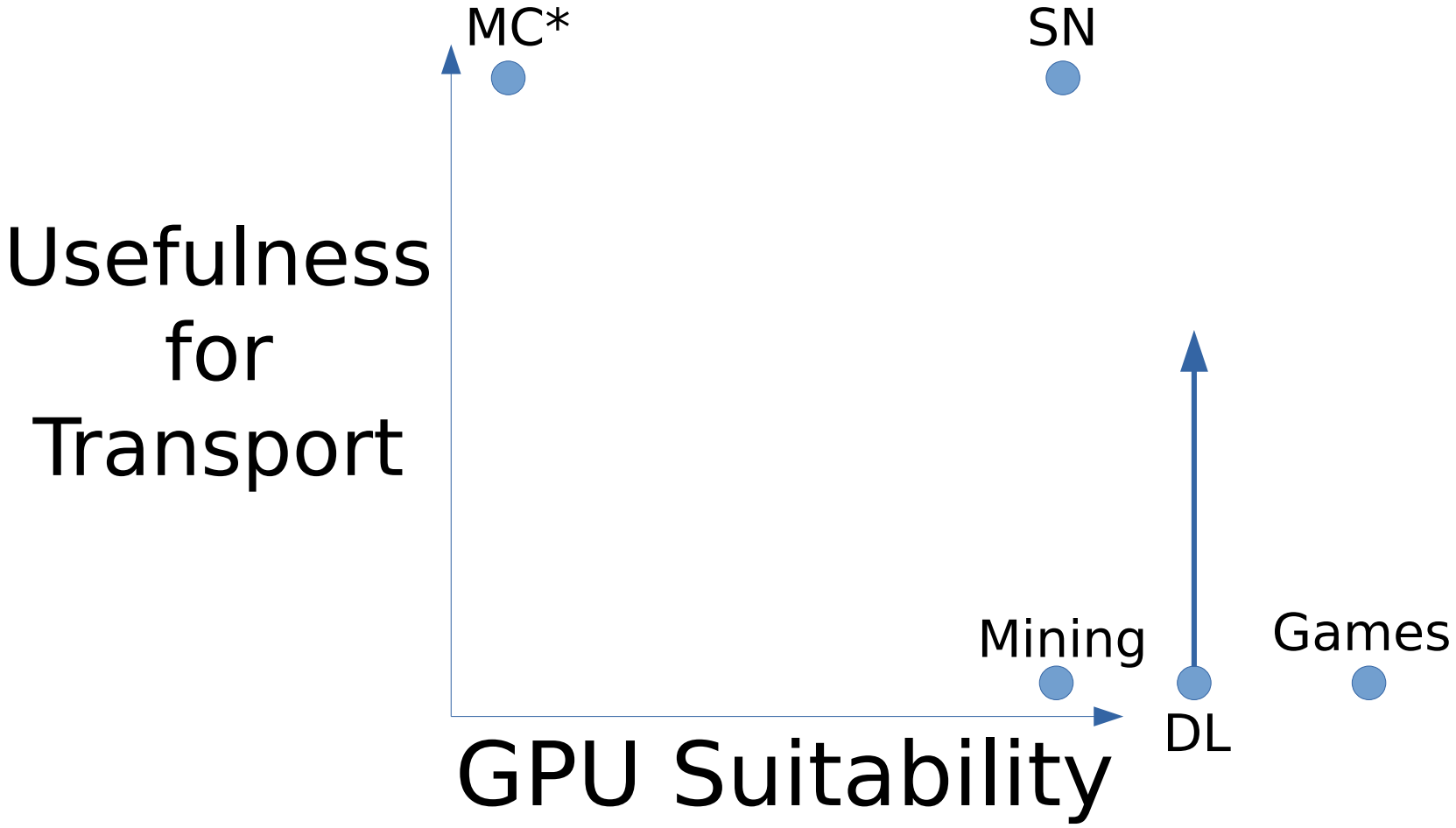
This talk has three parts

1) Motivation
2) Methods
3) Results
   - The transport equation can be solved using a neural network (NN).

Can we use what GPUs were designed for?

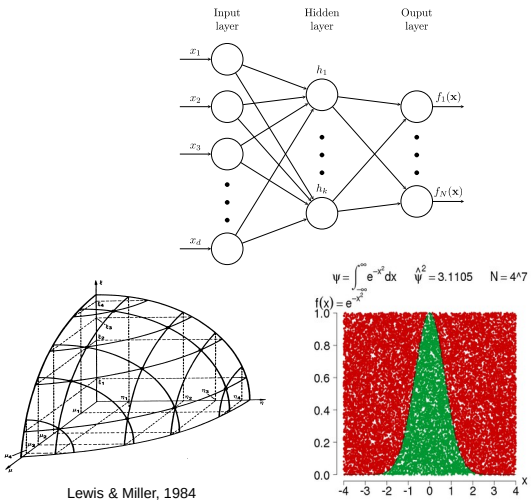


Can we use what GPUs were designed for?



**History-based* Monte Carlo

Can we improve previous results?



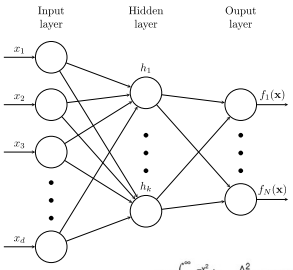
Lewis & Miller, 1984

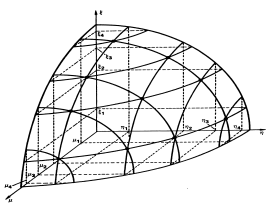
http://mike.pozulp.com/2019nnPaper.pdf

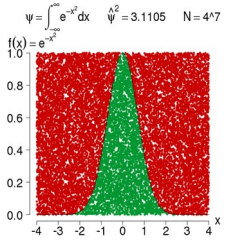
http://mike.pozulp.com/2019nnSlides.pdf

http://mike.pozulp.com/2019nnErrata.pdf

We added Het, Hyp, and GPUs





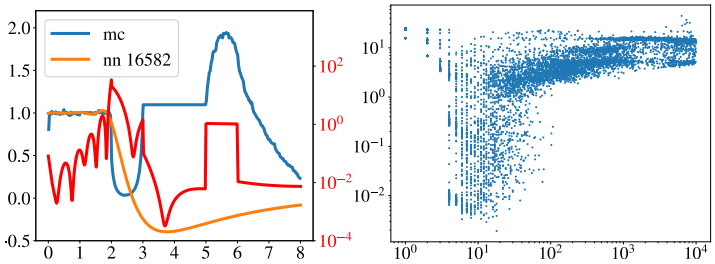


Lewis & Miller, 1984

http://mike.pozulp.com/2019nnPaper.pdf

http://mike.pozulp.com/2019nnSlides.pdf

http://mike.pozulp.com/2019nnErrata.pdf



Time(%)	Time	Calls
54.06	82.7777s	6127206
21.69	33.2049s	40848436
11.25	17.2308s	17997061
6.85	10.4897s	562472
1.90	2.91627s	7

http://mike.pozulp.com/2021hhg.pdf

https://github.com/llnl/narrows

A neural network is a function approximation technique

- Input $\mathbf{x} \in \mathbb{R}^d$
- Output $f(\mathbf{x}) \in \mathbb{R}^N$
- Non-linear activation function

$$h_i = \sigma(\mathbf{w}_i^T \mathbf{x} + b_i)$$

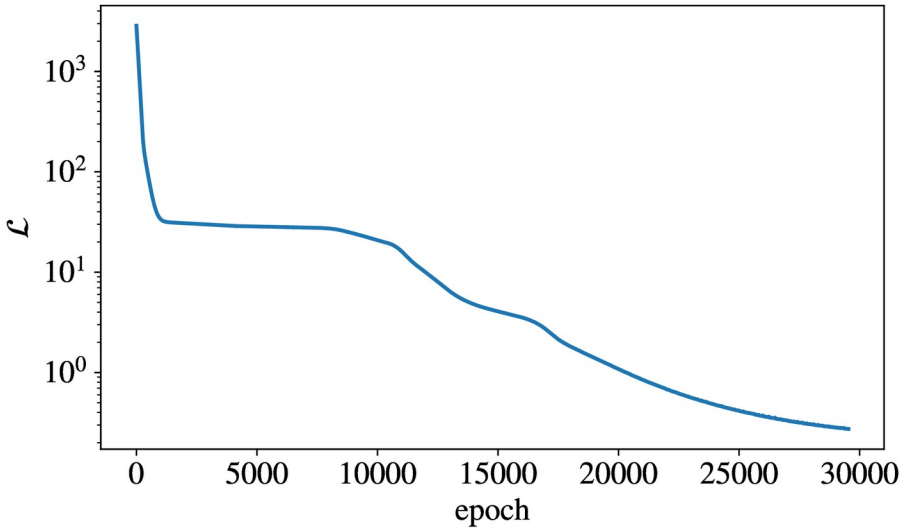
- Weights $\mathbf{w}_i \in \mathbb{R}^n$
- Biases b_i

The neural network minimizes a loss function

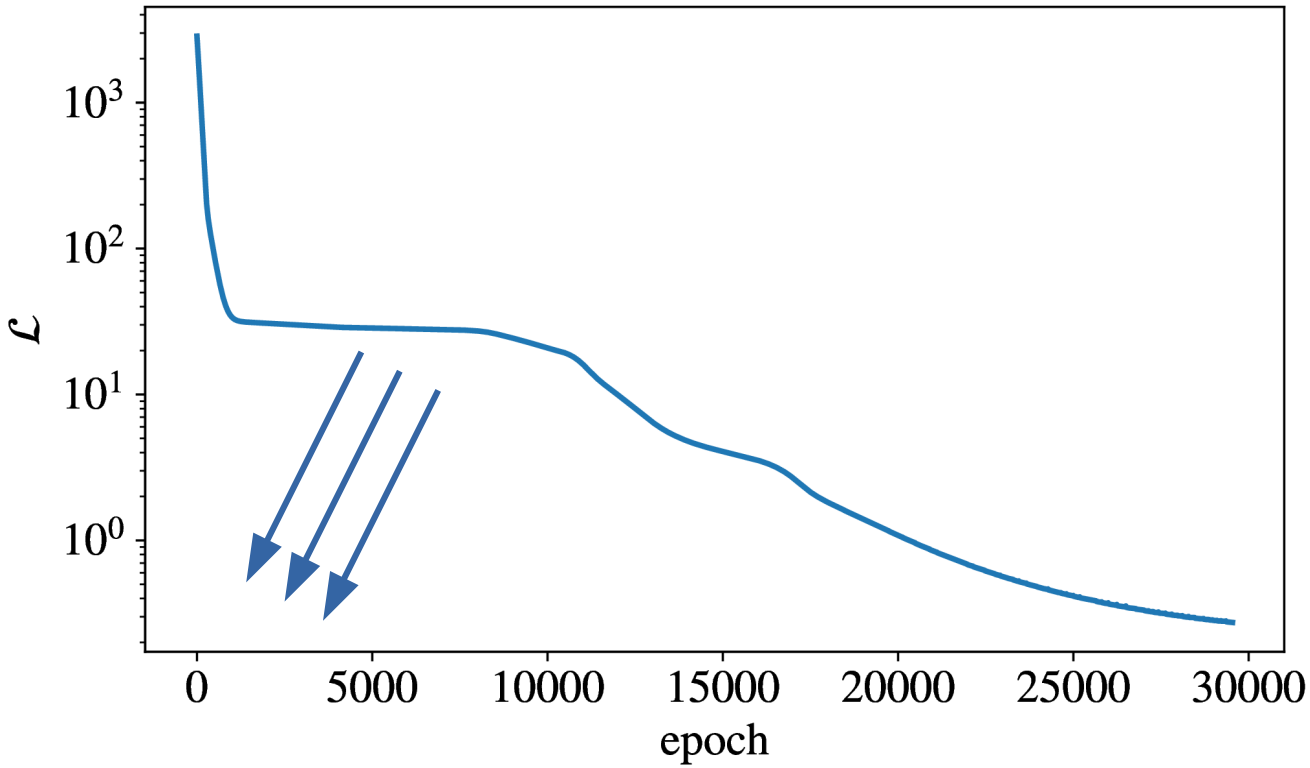
- Input $\mathbf{x} \in \mathbb{R}^d$
- Output $f(\mathbf{x}) \in \mathbb{R}^N$
- Non-linear activation function

$$h_i = \sigma(\mathbf{w}_i^T \mathbf{x} + b_i)$$

- Weights $\mathbf{w}_i \in \mathbb{R}^n$
- Biases b_i



Hyperparameters affect convergence



We solved a 1D mono-energetic transport equation with linearly anisotropic scattering

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

$$\psi(z = 0, \mu > 0) = 0 \quad (1b)$$

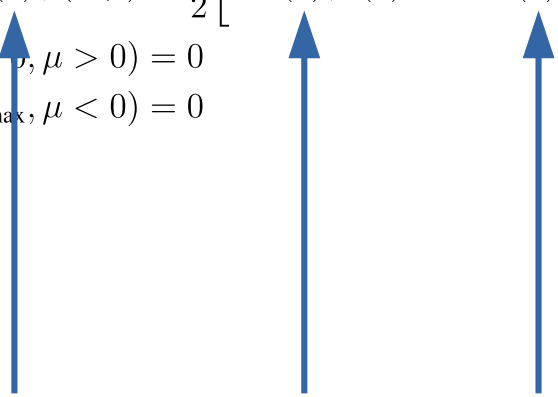
$$\psi(z = z_{\max}, \mu < 0) = 0 \quad (1c)$$

Heterogeneity means multi-material

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

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

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



We used this loss function

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

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

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

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

We solved for scalar flux, current

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

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

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

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

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

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

We used this convergence criterion

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

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

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

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

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

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

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

We used this NN architecture

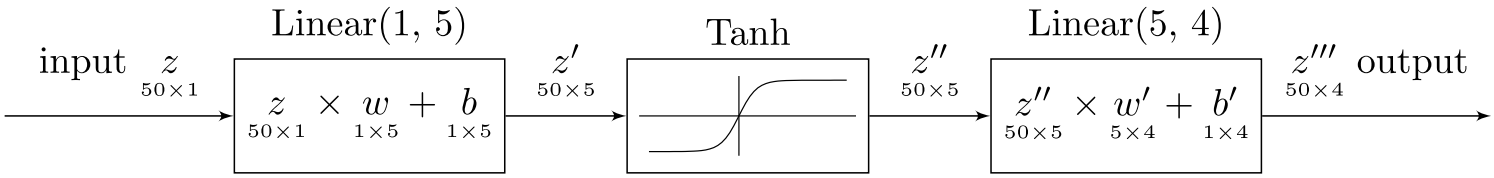


Figure 1: Default NN architecture.

$$p(h, N) = (h + h) + (hN + N)$$

p	number of parameters
h	number of hidden layer nodes
N	number of ordinates

$$p(5, 4) = 34$$

We used a 6-step procedure

1. Construct the NN and define the loss function to minimize.

We used a 6-step procedure

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 $\hat{\mathbf{\Phi}}_1$.

We used a 6-step procedure

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).

We used a 6-step procedure

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.

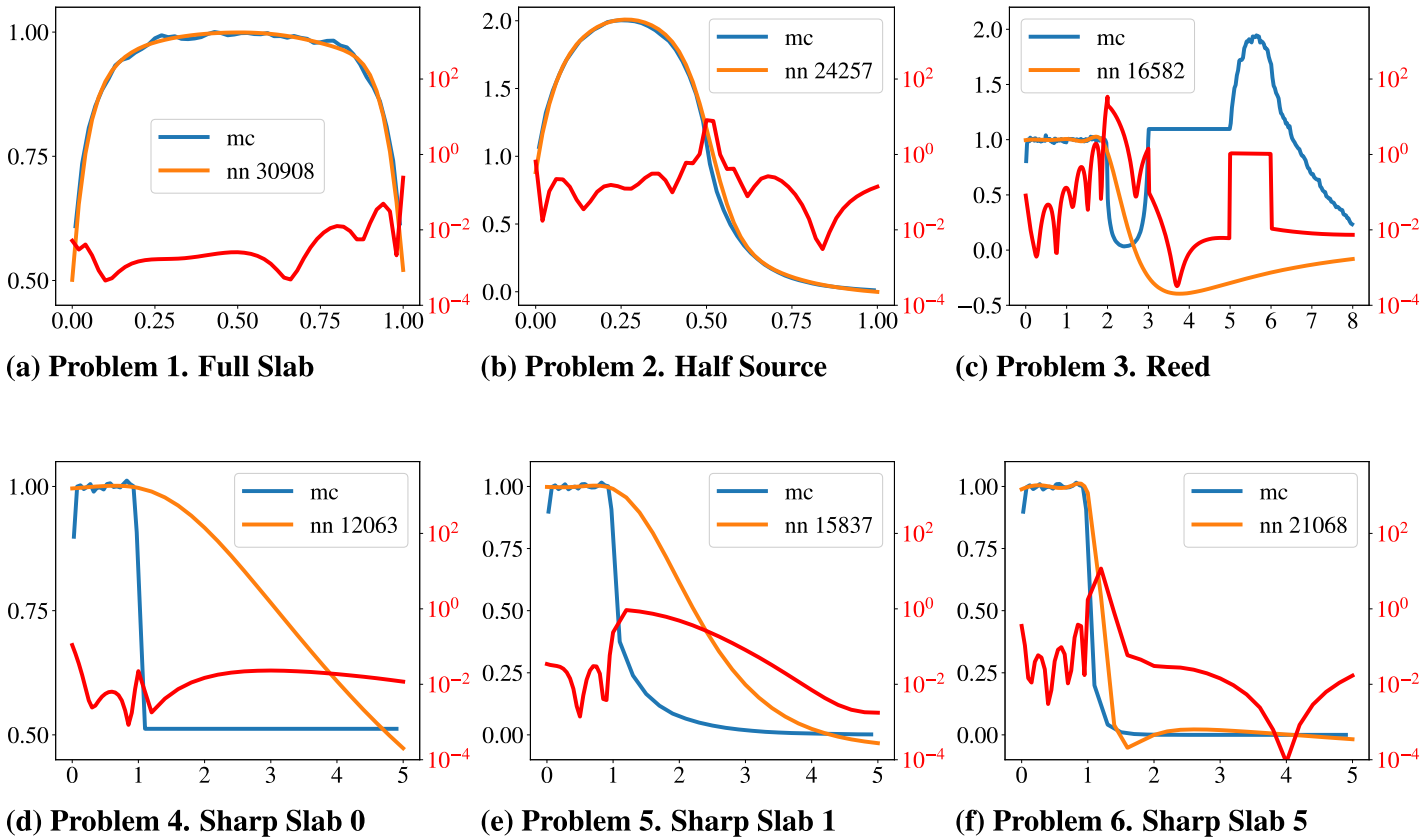
We used a 6-step procedure

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.

We used a 6-step procedure

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.

We ran six problems

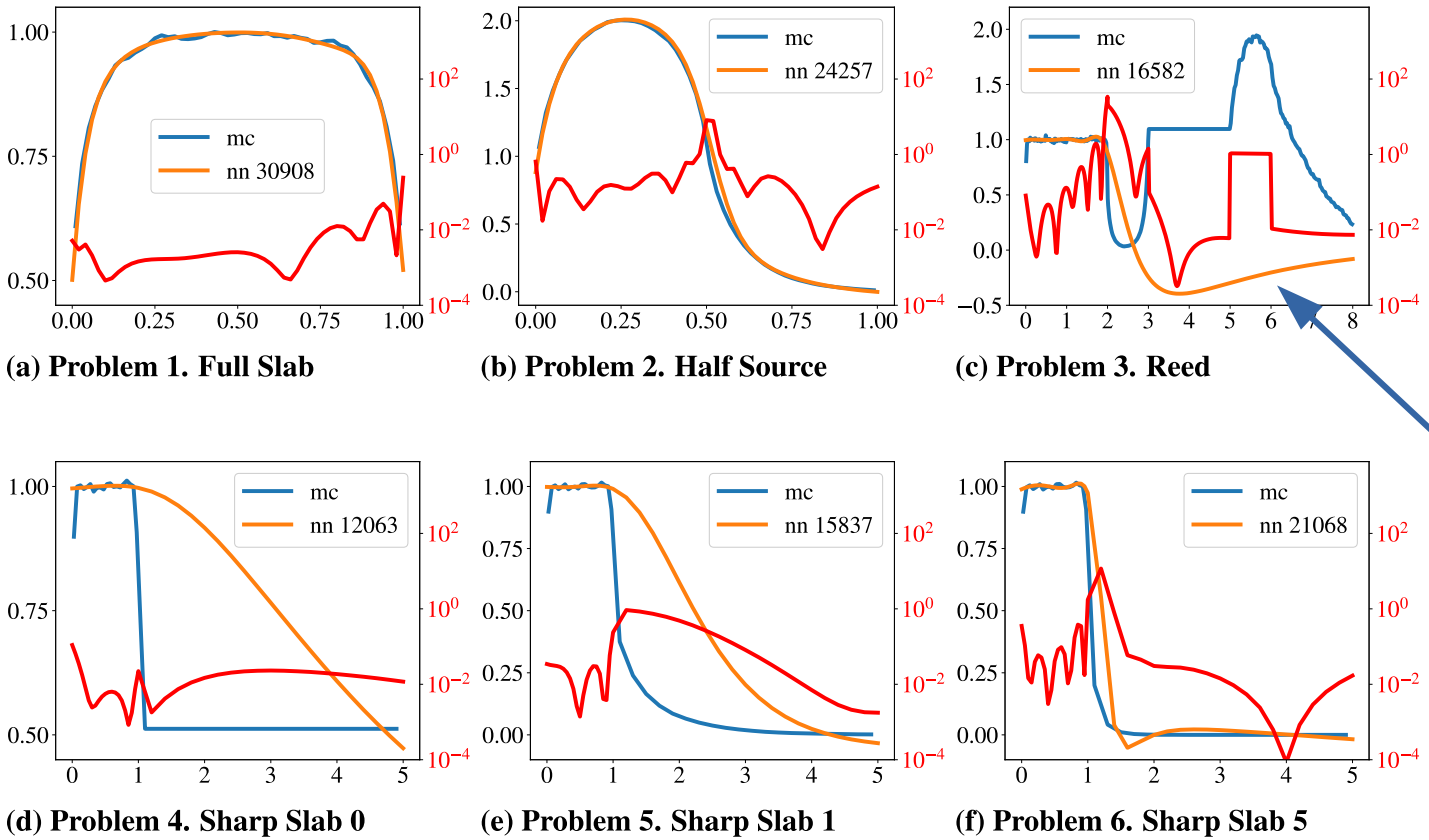


(a) Problem 1. Full Slab (b) Problem 2. Half Source (c) Problem 3. Reed

(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.

We made progress in understanding NN accuracy for heterogeneous problems but failed to accurately solve the most heterogeneous problem

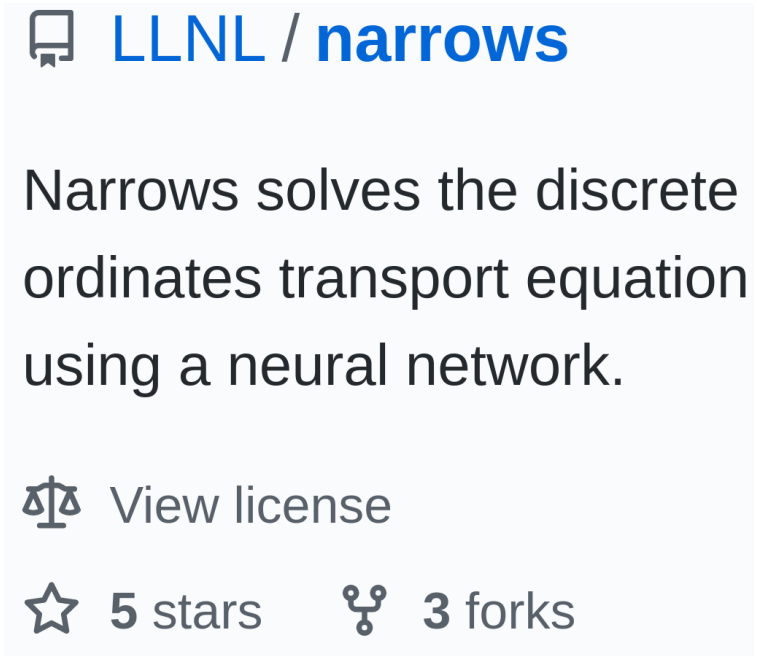


(a) Problem 1. Full Slab (b) Problem 2. Half Source (c) Problem 3. Reed

(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.

But we released our code...



github.com/llnl/narrows

...and Ravi Kumar solved it!

LLNL / narrows

Narrows solves the discrete ordinates transport equation using a neural network.

View license

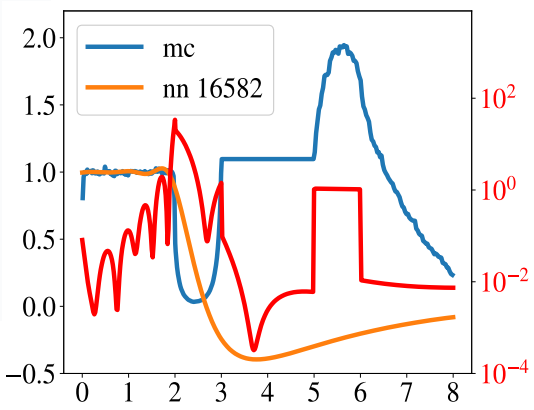
5 stars 3 forks

github.com/llnl/narrows

A Study of Artificial Neural Networks for the Solution of Multi-group Slab Geometry Discrete Ordinates Transport Problems with Material Heterogeneity

by

Ravi Kumar



(c) Problem 3. Reed

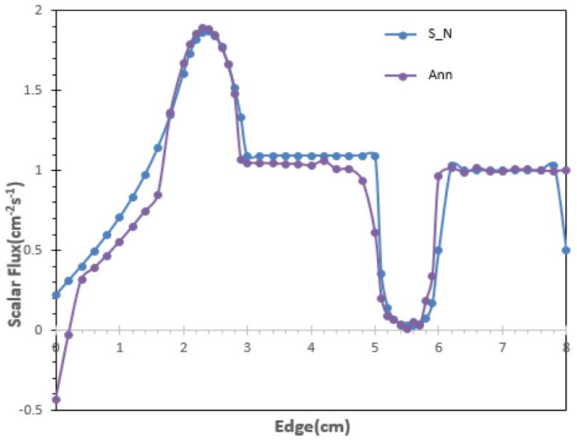


Figure 14: Scalar flux comparison for Reed's problem.

ir.library.oregonstate.edu/concern/graduate_thesis_or_dissertations/zs25xg841

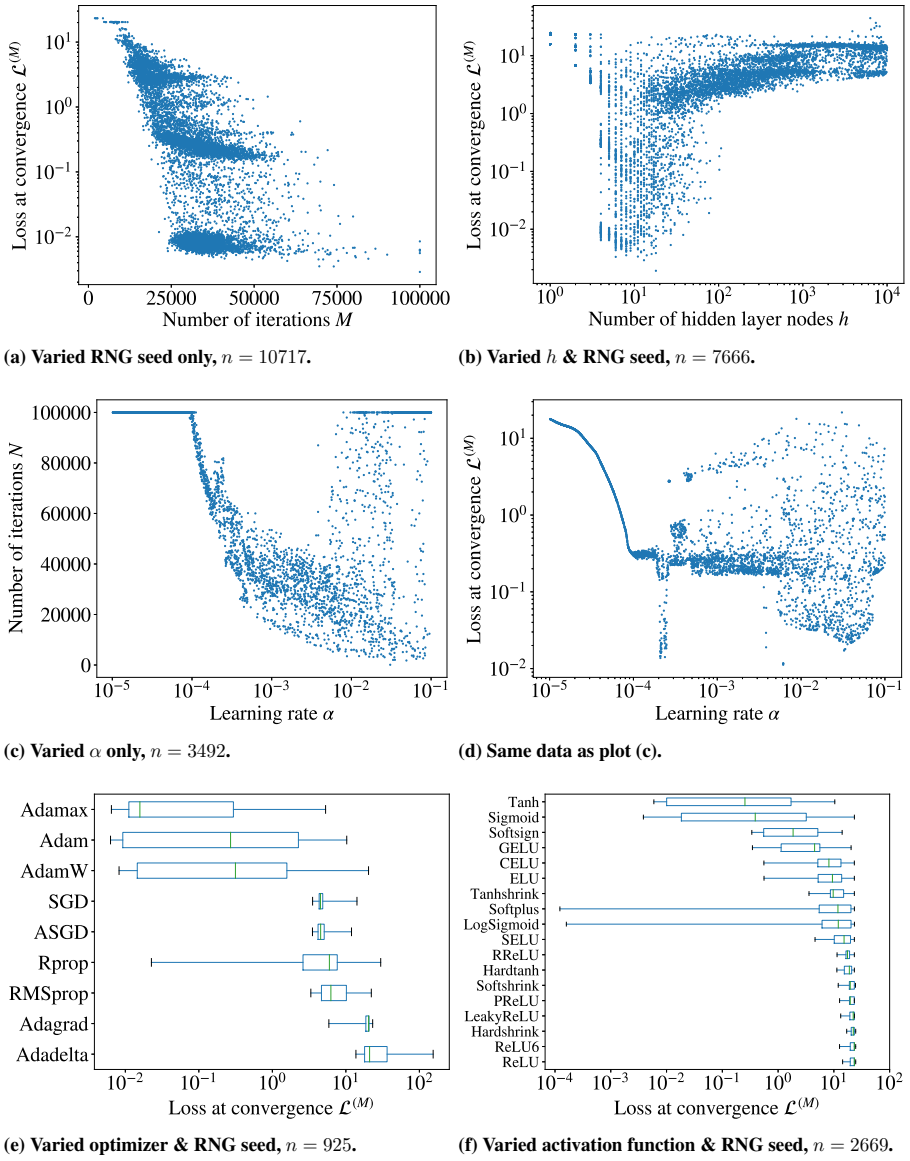
Kumar went deeper than we did

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	

Hyp study found sensitivities to

- RNG seed (a-f)



(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).

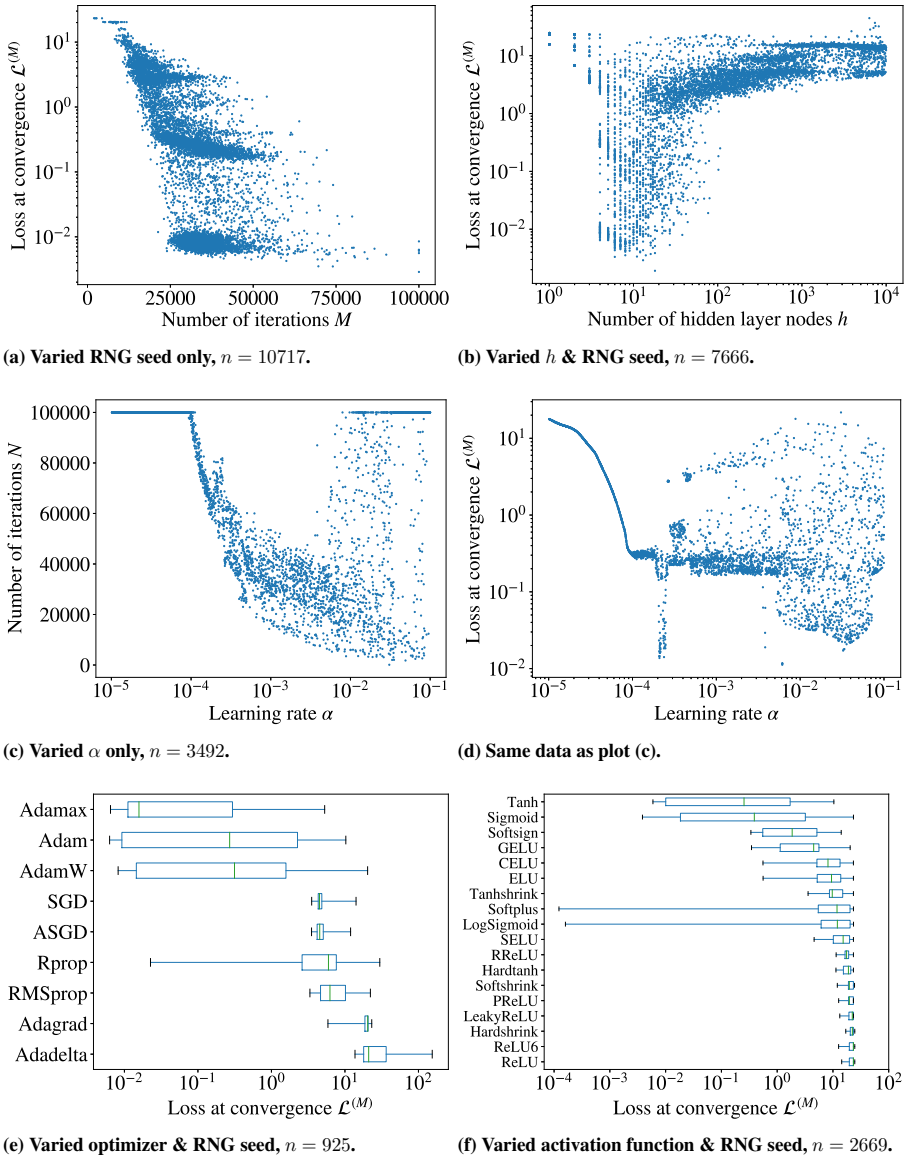
(e) Varied optimizer & RNG seed, $n = 925$.

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

Figure 3: Hyperparameter study results for Problem 1.

Hyp study found sensitivities to

- RNG seed (a-f)
- Number of hidden layer nodes (b)



(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).

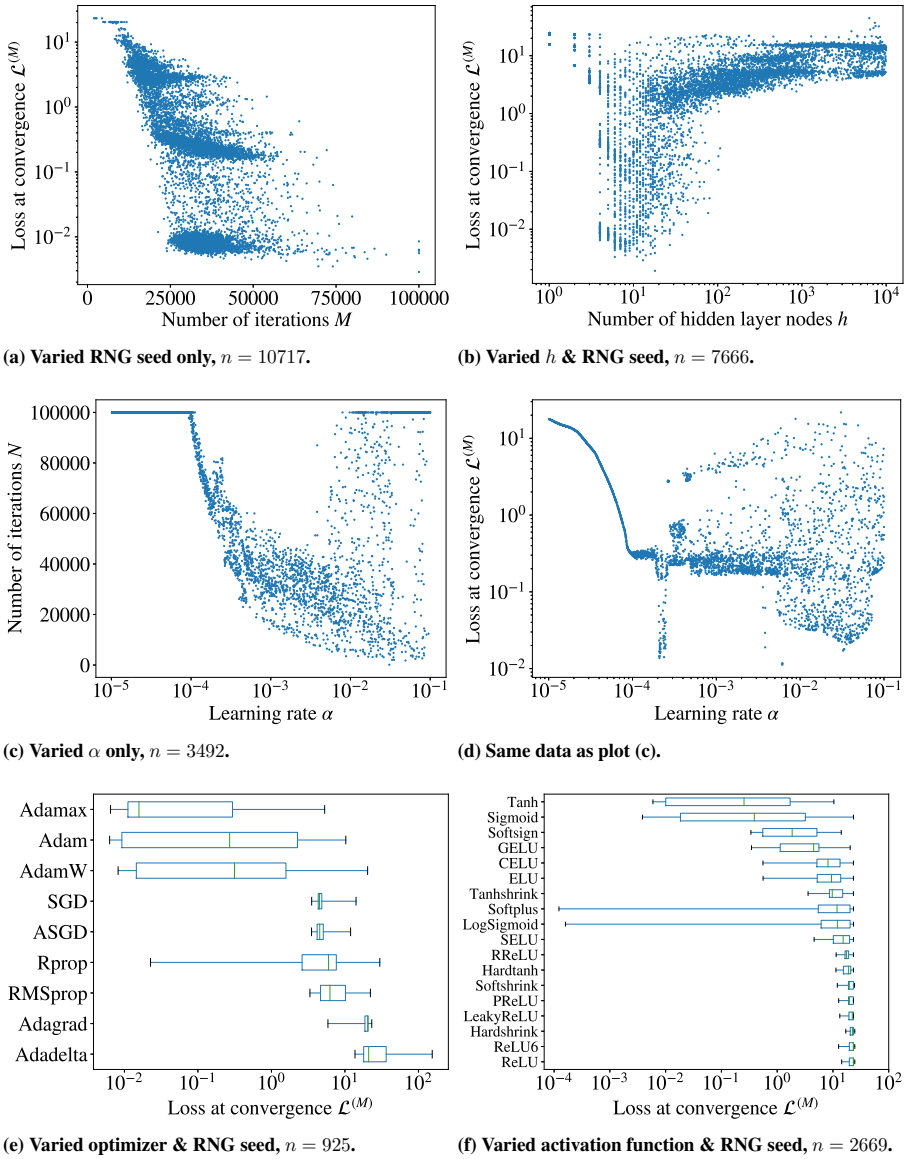
(e) Varied optimizer & RNG seed, $n = 925$.

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

Figure 3: Hyperparameter study results for Problem 1.

Hyp study found sensitivities to

- RNG seed (a-f)
- Number of hidden layer nodes (b)
- Learning rate (c-d)



(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).

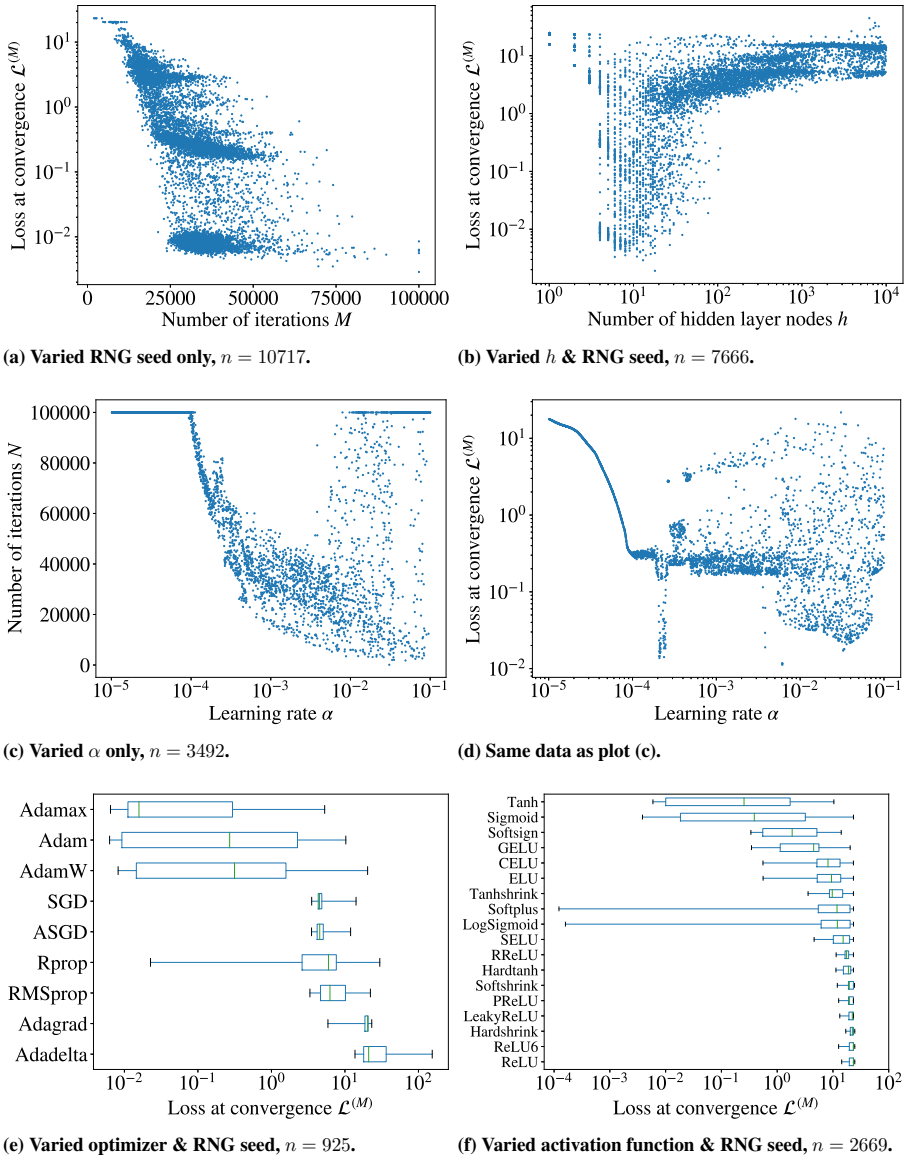
(e) Varied optimizer & RNG seed, $n = 925$.

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

Figure 3: Hyperparameter study results for Problem 1.

Hyp study found sensitivities to

- RNG seed (a-f)
- Number of hidden layer nodes (b)
- Learning rate (c-d)
- Optimizer (e)



(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).

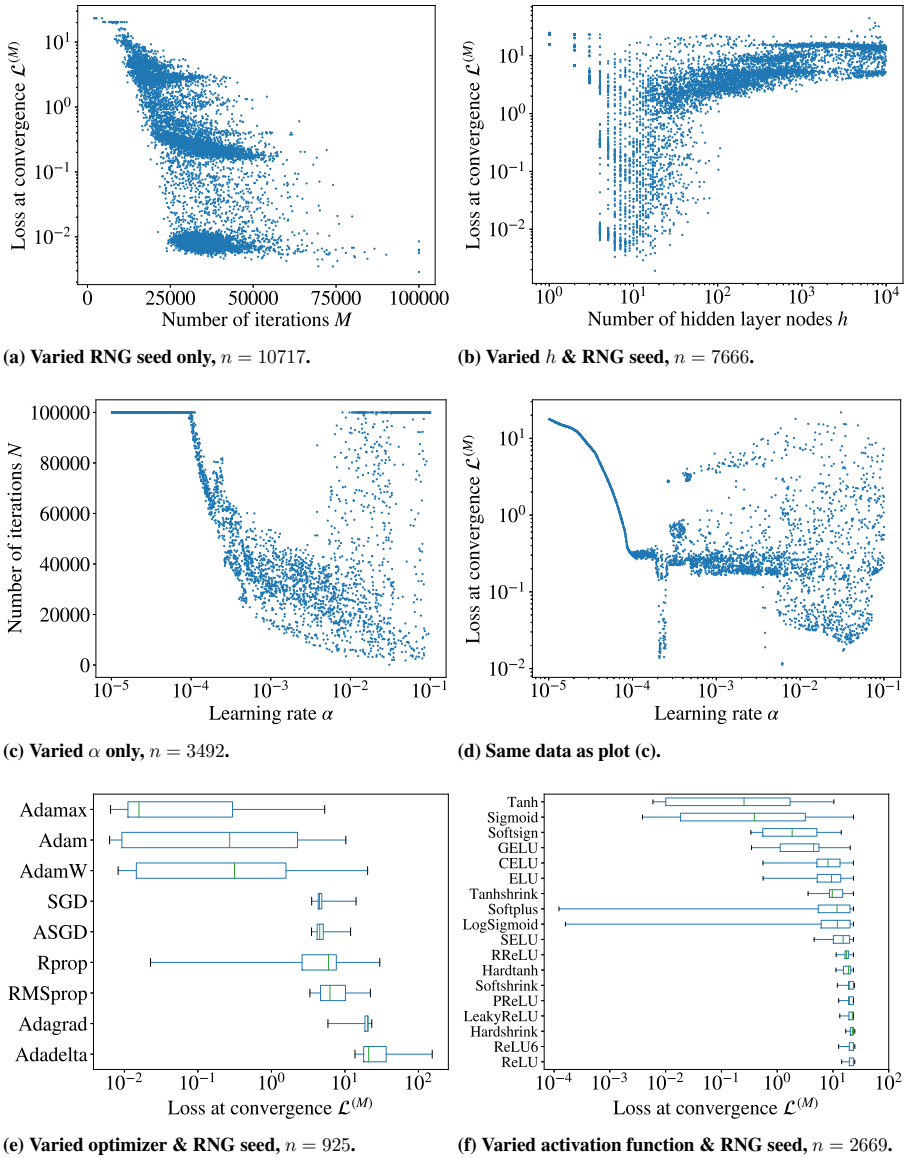
(e) Varied optimizer & RNG seed, $n = 925$.

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

Figure 3: Hyperparameter study results for Problem 1.

Hyp study found sensitivities to

- RNG seed (a-f)
- Number of hidden layer nodes (b)
- Learning rate (c-d)
- Optimizer (e)
- Activation function (f)



(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.

We observed a GPU slowdown

	Runtime	Slowdown
1 P9 core	2 min 3 sec	1x
V100 GPU	5 min 44 sec	2.8x

A bigger network could help

	Runtime	Slowdown
1 P9 core	2 min 3 sec	1x
V100 GPU	5 min 44 sec	2.8x

- NN is serial w.r.t. optim iters

  => available concurrency limited by:

  - Number of parameters
  - Number of training data points

AlexNet >> Narrows

	Runtime	Slowdown
1 P9 core	2 min 3 sec	1x
V100 GPU	5 min 44 sec	2.8x

- NN is serial w.r.t. optim iters

  => available concurrency limited by:

  - Number of parameters
  - Number of training data points

	Parameters	Training data
Narrows	34	50 scalars
AlexNet	60 million	1.2 million images

Kernel launch and runtime API overheads dominate execution

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

We have action items

Action items

- Add depth
  - Verify Kumar
- Run big (S32, 10k)
  - Reduce GPU slowdown

We have a wish list

Action items

- Add depth
  - Verify Kumar
- Run big (S32, 10k)
  - Reduce GPU slowdown

Wish list

- Performance
- Multigroup
- 2D
- Conservation
- Symmetry preservation

These are the paper authors

Action items

- Add depth
  - Verify Kumar
- Run big (S32, 10k)
  - Reduce GPU slowdown

Wish list

- Performance
- Multigroup
- 2D
- Conservation
- Symmetry preservation




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.