1D Transport Using Neural Nets, S_N, and MC

Michael M. Pozulp
University of California, Davis, and
Lawrence Livermore National Laboratory
7000 East Ave, L-170
Livermore, CA 94550
pozulp1@llnl.gov

ABSTRACT

A method previously introduced to solve the discrete ordinates neutron transport equation (NTE) in a slab geometry using a neural network (NN) is revisited. A runtime comparison shows that the NN method can be faster than S_N and MC while achieving the same accuracy for an example problem.

KEYWORDS: neural network, machine learning, 1D transport

1. INTRODUCTION

The field of *machine learning* (ML) traditionally applies mathematical models to data as a means of solving problems such as pattern recognition, but other problems are also solvable with ML. Burgers' equation, Schrödinger's equation [1], and Laplace's equation [2] can each be posed as ML problems and solved using a function approximation technique known as a *neural network* (NN) [3]. Bespoke ML hardware [4,5], made to run highly-tuned ML software [6–8], can solve equations faster, as long as those equations can be posed as ML problems. Here, a previously introduced method using NNs [9] is implemented. The current work differs from [9] by reporting the runtime of NN training and prediction, comparing it to S_N and MC runtimes, and observing that NN prediction is faster than SN and MC while providing equivalent accuracy.

2. METHODS

2.1. Slab Geometry Neutron Transport

Consider the slab geometry NTE, which is a function of a single spatial variable z and directional variable μ [10]

$$\mu\frac{\partial\psi(z,\mu)}{\partial z} + \Sigma_t(z)\psi(z,\mu) = 2\pi\int_{-1}^{1} d\mu'\Sigma_s(z,\mu_0)\psi(z,\mu') + \frac{1}{2}\left[\nu\Sigma_f(z)\phi(z) + Q_{ext}(z)\right] \quad (1)$$

Section 2.2 describes the use of a discrete ordinates approach to solve Equation (1), Section 2.4 describes the use of a neural network approach, and Section 2.5 describes a Monte Carlo approach.

2.2. Solutions using the Discrete Ordinates Method

An even-order Gauss-Legendre quadrature set is employed and the spatial variable is discretized using the finite volume method. The resulting discrete ordinates (S_N) equations are [11]

$$\frac{\mu_m}{l_j}\left(\psi^{(l+1)}_{m,j+\frac{1}{2}} - \psi^{(l+1)}_{m,j-\frac{1}{2}}\right) + \Sigma_{t,j}\psi^{(l+1)}_{m,j} = \frac{1}{2}\hat{Q}^{(l)}_{m,j} \tag{2}$$

where

$$\hat{Q}^{(l)}_{m,j} = \Sigma_{s0,j}\Phi^{(l)}_{0,j} + 3\mu_m\Sigma_{s1,j}\Phi^{(l)}_{1,j} + Q_{m,j} \tag{3}$$

$$\Phi^{(l+1)}_{0,j} \equiv \sum_{m=1}^{N}\psi^{(l+1)}_{m,j}w_m \tag{4}$$

$$\Phi^{(l+1)}_{1,j} \equiv \sum_{m=1}^{N}\mu_m\psi^{(l+1)}_{m,j}w_m \tag{5}$$

$$\psi^{(l+1)}_{m,j} = \frac{1+\alpha_{m,j}}{2}\psi^{(l+1)}_{m,j+\frac{1}{2}} + \frac{1-\alpha_{m,j}}{2}\psi^{(l+1)}_{m,j+\frac{1}{2}} \tag{6}$$

with boundary conditions

$$\psi^{(l+1)}_{m,\frac{1}{2}} = f_m \qquad \mu_m > 0 \tag{7}$$

$$\psi^{(l+1)}_{m,J+\frac{1}{2}} = g_m \qquad \mu_m < 0 \tag{8}$$

Convergence is determined by the relative pointwise convergence criterion

$$\max_{1\le j\le J}\left|\frac{\phi^{(l+1)}_j - \phi^{(l)}_j}{\phi^{(l)}_j}\right| < \epsilon \tag{9}$$

The S_N method is well-known [12] and is employed for production use in industry and academia.

2.3. Artificial Neural Networks

Artificial Neural Networks (ANNs) are a class of machine learning algorithms which provide a general means for non-linear function approximation. That is, ANNs can approximate an arbitrary function $f(\mathbf{x})$ for some input $\mathbf{x}$.

The general procedure for approximating functions using ANNs is to perform a series of operations on an input $\mathbf{x}$, which ultimately result in the output $f(\mathbf{x})$. The operations performed on the data $\mathbf{x}$ are specified by the user, and are collectively referred to as the *architecture* of the ANN. ANN architectures consist of *layers* which define the operations used. In general, ANNs consist of an input layer, one or more hidden layers, and an output layer. The term *deep learning* is a reference to this layer structure as deep learning architectures are several layers deep. Each of these layers is connected in some way, and between them often exist *activation functions*, which may be

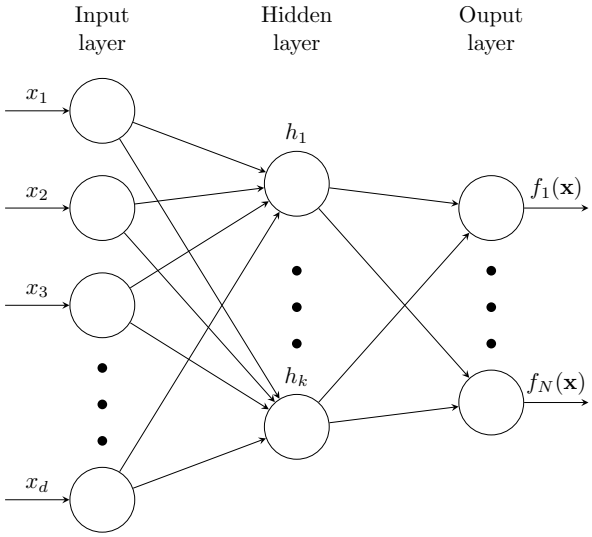

Figure 1: Schematic of an ANN with a single fully-connected hidden layer.

nonlinear. For a layer to be *fully-connected*, each element of the layer must be connected to each element of the following layer.

Consider an ANN that takes as input $\mathbf{x} \in \mathbb{R}^d$ and has a single fully-connected layer with k nodes, and outputs $f(\mathbf{x}) \in \mathbb{R}^N$, as shown in Figure 1. The connections between subsequent layers is represented as a matrix product between values of nodes and parameters of the network. In this example, the value at node i, v_i, is calculated as $v_i = \mathbf{w}_i\mathbf{x} + b_i$, where $\mathbf{w}_i$ is a vector of *weights* and b_i is a *bias*. The value at each node can then be passed to the activation function, yielding $h_i = \sigma(\mathbf{w}_i x + b_i)$. In vector form, the hidden layer is then $\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$, where $\mathbf{W}$ is a matrix with the ith row being the weights $\mathbf{w}_i$. The quantity $\mathbf{h}$ can then be sent to the output layer as another set of operations, such that

$$\mathbf{h}_2 = \mathbf{W}_2\mathbf{h} + \mathbf{b}_2 = \mathbf{W}_2\sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) + \mathbf{b}_2 = f(\mathbf{x}) \tag{10}$$

Within this architecture, there are a number of tunable parameters, namely the weights $\mathbf{w}_i$ and biases b_i, which determine the accuracy in the approximation $f(\mathbf{x}) = y$ for some known value of y. The accuracy with which $f(\mathbf{x})$ approximates y is measured by a *loss function*. One error measure that can be used in a loss function is the mean squared error (MSE):

$$\text{MSE} = \frac{1}{n}\|\mathbf{y} - f(\mathbf{x})\|^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2 \tag{11}$$

where the vectors $\mathbf{y}$ and $f(\mathbf{x})$ refer to a collection of n samples. The loss function is minimized, meaning that $f(\mathbf{x})$ is approximated as the true value y, by tuning the parameters of the ANN, a process referred to as *training*. The training procedure consists of finding the parameters $\mathbf{W}, \mathbf{b}, \mathbf{W}_2, \mathbf{b}_2$, collectively referred to as the parameters $\hat{\mathcal{P}}$, such that

$$\hat{\mathcal{P}} = \arg\min_{\mathcal{P}}\{\text{MSE}\} \tag{12}$$

In minimizing this objective function, or loss, optimization schemes such as stochastic gradient descent [3] are used, which updates parameters in a direction that causes the objective function to decrease.

Once an architecture is specified, the procedure of training an ANN then consists of a *forward pass* in which $\mathbf{x}$ is propagated through the network to yield the estimate $f(\mathbf{x})$, followed by a *backward pass* in which gradients are computed at each layer and propagated through the network. The passing of gradients backwards through the network is referred to as *backpropagation* or *backprop* for short, and relies on the chain rule from calculus to efficiently compute gradients without performing redundant calculations. This procedure of forward and backward passes through the network is often referred to as an *epoch*, and is repeated many times, often until the loss function (i.e., MSE) converges to some specified tolerance.

2.4. Solving the S_N Equation using ANNs

The system from Equation (1) can be solved using an ANN. In particular, optimal network parameters $\hat{\mathcal{P}}$ that yield the lowest error in the solution to Equation (1) are found. This problem was previously considered by [9]. The method in [9] is implemented using a modern neural network framework, namely the `PyTorch` [8] deep learning framework in `Python`.

First, the network architecture in [9] is reproduced, which is a single fully-connected layer, with a hyperbolic tangent ($\tanh$) activation function leading to the output layer (i.e., $\sigma(\cdot) = \tanh(\cdot)$). Consider the set of S_N equations for the slab geometry angular flux $\boldsymbol{\Psi}(\mathbf{z}) = \boldsymbol{\Psi} \in \mathbb{R}^{d\times N}$ sampled at spatial points $\mathbf{z} \in \mathbb{R}^d$ [11]

$$\nabla\boldsymbol{\Psi}\text{diag}(\boldsymbol{\mu}) + \Sigma_t\boldsymbol{\Psi} = \frac{1}{2}\big(\Sigma_{s0}\boldsymbol{\Phi}_0\mathbb{1}_N^T + 3\Sigma_{s1}\boldsymbol{\Phi}_1\boldsymbol{\mu}^T\big) + \frac{1}{2}\mathbf{Q} \tag{13}$$

where $\text{diag}(\boldsymbol{\mu}) \in \mathbb{R}^{N\times N}$ is a matrix with ordered values of the ordinate angles μ_i on the diagonals, and $\mathbb{1}_N^T$ is a row vector of N ones. Note that the spatial argument $\mathbf{z}$ has been dropped from the scalar and angular fluxes for brevity. The neural network is used to estimate $\hat{\boldsymbol{\Psi}}$, allowing us to compute $\hat{\boldsymbol{\Phi}}_0$ and $\hat{\boldsymbol{\Phi}}_1$ as

$$\hat{\boldsymbol{\Phi}}_0 = \hat{\boldsymbol{\Psi}}\mathbf{w} \tag{14}$$

$$\hat{\boldsymbol{\Phi}}_1 = \hat{\boldsymbol{\Psi}}\text{diag}(\boldsymbol{\mu})\mathbf{w} \tag{15}$$

where $\mathbf{w} \in \mathbb{R}^N$ are the Gauss-Legendre quadrature weights.

The error can then be measured as how well the ANN solution satisfies Equation (13) using the squared error:

$$\begin{aligned}\mathcal{L} = &\left\|\nabla\hat{\boldsymbol{\Psi}}\text{diag}(\boldsymbol{\mu}) + \Sigma_t\hat{\boldsymbol{\Psi}} - \frac{1}{2}\big(\Sigma_{s0}\hat{\boldsymbol{\Phi}}_0\mathbb{1}_N^T - 3\Sigma_{s1}\hat{\boldsymbol{\Phi}}_1\boldsymbol{\mu}^T - \mathbf{Q}\big)\right\|_F^2 \\ &+ \gamma_L\|\hat{\boldsymbol{\Psi}}^{\mu>0}(z=0) - \boldsymbol{\Psi}_L\|_F^2 \\ &+ \gamma_R\|\hat{\boldsymbol{\Psi}}^{\mu<0}(z=z_{max}) - \boldsymbol{\Psi}_R\|_F^2\end{aligned} \tag{16}$$

where F denotes the Frobenius norm. The last two terms in Equation (16) act as regularizers to enforce specific values at the boundaries (i.e., the left (L) and right (R) boundary conditions) and

each $\gamma_{\{L,R\}}$ is a regularization coefficient. The training procedure consists of finding the neural network parameters such that $\hat{\mathbf{\Psi}}$ mininizes the loss in Equation (16).

In most ANN use cases, both the input data $\mathbf{x}$ and an associated output $y = f(\mathbf{x})$ are used in the training, a practice referred to as *supervised learning*. In this case, however, there is no labelled output y, and instead ANN parameters are found that minimize the loss accrued by performing operations on the output.

The *Adam* optimizer [13] is used to minimize the loss function in Equation (16). The Adam optimization routine updates weights $\mathbf{w}_i$ and biases b_i in a similar fashion to traditional stochastic gradient descent, however, it also adaptively updates step sizes which typically results in faster convergence than standard stochastic gradient descent.

The procedure for solving the slab geometry transport equation in Equation (13) is summarized as follows:

1. Construct the ANN and define a loss function to minimize.
2. Pass input data $\mathbf{z}$ through the network to yield $f(\mathbf{z})$.
3. Calculate the loss using Equation (16).
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 using 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 loss converges to a specified error tolerance.
7. Using this trained network, predict the angular flux $\hat{\mathbf{\Psi}}$ as well as the scalar flux $\hat{\mathbf{\Phi}}_0$.

The convergence criterion employed for training the neural network is

$$\left|\mathcal{L}^{(l+1)} - \mathcal{L}^{(l)}\right| < \epsilon \tag{17}$$

where $\mathcal{L}^{(l)}$ and $\mathcal{L}^{(l+1)}$ are the losses computed at epoch l and $l+1$, respectively and ϵ is some small constant (e.g., 10^{-10}).

2.5. Solutions using the Monte Carlo Method

Unlike the S_N solution to Equation (1), which requires solving a system of coupled linear equations, the Monte Carlo (MC) solution determines particle histories by pseudo-random sampling. All sampling employs the inverse-CDF technique [14]. The linearly anisotropic scattering outgoing angle and the distance to collision are sampled according to [15] and [16]. The MC method is well understood [12], and is employed for production use in industry and academia.

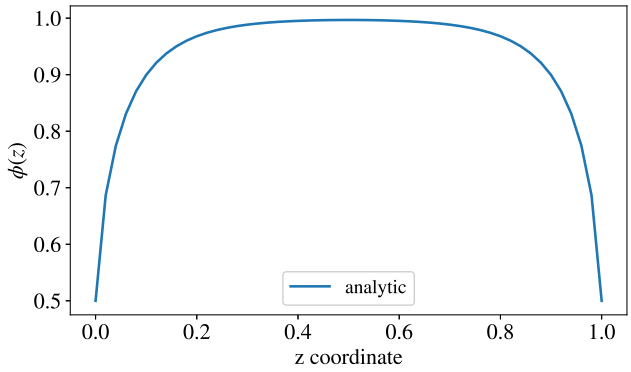

Figure 2: Analytic solution for the scalar flux $\phi(z)$ in Problem 1, given by Equation (18).

2.6. Model Comparison

These methods are compared by solving for the scalar flux $\phi(z)$ in a single slab of a completely absorbing material (i.e., $\Sigma_t = \Sigma_a$, $\Sigma_{s\{0,1\}} = 0$, $\Sigma_f = 0$). The slab is 1 cm in length, with vacuum boundary conditions (i.e., the angular flux coming from outside of the slab is zero) and has an external source uniformly distributed through the material. In this problem there is an exact solution for the scalar flux $\phi(z)$, shown in Figure 2 [17]

$$\phi(z) = \frac{Q_0}{\Sigma_a} - \frac{Q_0}{2\Sigma_a}\left(e^{-\Sigma_a z} + e^{-\Sigma_a(H-z)}\right) + \frac{Q_0}{2\Sigma_a}\left(zE_1(\Sigma_a z) + (H-z)E_1(\Sigma_a(H-z))\right) \quad (18)$$

where $z \in [0, H]$.

The three algorithms are compared using max relative error

$$r_m = \max_j \frac{|\phi_j - \hat{\phi}_j|}{\phi_j} \quad (19)$$

where $m \in \{\text{nn, sn, mc}\}$ denotes the different algorithms, $\hat{\phi}_j$ is the model solution at spatial point j, and ϕ_j is the analytic solution at spatial point j.

3. RESULTS

The implementation [18] of all three solution methods used the parameters given in Table 1 and was run on a 2.3 GHz Intel Core i5 2410M processor. Table 2 shows a comparison of the error between the various methods, as well as their runtime. From this table, one can see that the NN achieved near-equal accuracy with S_N, but at four orders of magnitude greater runtime. Out of the three methods, Monte Carlo was the most accurate.

Figure 3 shows the NN convergence. The loss function decreased quickly until epoch 1,000,

where it remainted relatively constant until epoch 9,000, after which it decreased steadily. The convergence criterion ϵ_{nn} was achieved just before epoch 30,000.

Table 1: Parameters used to specify and solve the uniform source problem.

Parameter	Value	Description
Σ_t	8	Total cross section (cm^{-1})
Σ_{s0}	0	Total scattering cross section (cm^{-1})
Σ_{s1}	0	Linearly anisotropic cross section (cm^{-1})
Q_0	8	External source magnitude
J_{nn}	50	Number of zones for NN solution
J_{sn}	50	Number of zones for S_N solution
J_{mc}	50	Number of zones for MC solution
NR_{nn}	4	Number of ordinates for NN solution
NR_{sn}	4	Number of ordinates for S_N solution
NP	1e6	Number of particles for MC solution
ϵ_{nn}	1e-6	Convergence criterion value for NN solution
ϵ_{sn}	1e-13	Convergence criterion value for S_N solution
k	5	Number of hidden layer nodes in NN

Table 2: Max relative error, its location, and runtimes for Problem 1.

Algorithm	Max. rel. err.	z (cm)	Runtime (s)
nn train	-	-	6.97e+01
nn pred	0.051807	0.98	1.39e-04
sn	0.047869	0.02	4.39e-03
mc	0.013654	0.01	2.77e+00

After training a network using the number of zones listed in Table 1, the solution was predicted at different zone counts. Figure 4 shows the relative error as a function of number of zones. From this figure, one can see that the NN prediction was more accurate than S_N at 10 zones. The accuracy for all other zone counts is approximately equal for both the NN prediction and S_N.

Figure 5 shows the time to solution for NN prediction and S_N as a function of number of zones. This figure shows that for all zone counts except ten, the NN *prediction* was almost two orders of

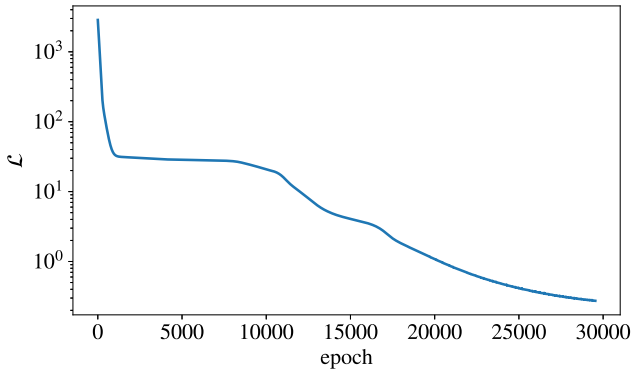

Figure 3: NN loss by epoch during training for Problem 1.

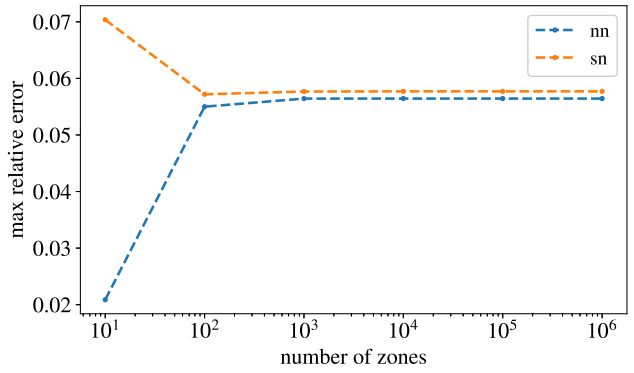

Figure 4: Maximum relative error for predictions by zone count for Problem 1.

magnitude faster than S_N. On graphics processing units (GPUs), tensor cores, and tensor processing units, NN predictions are expected to be more than two orders of magnitude faster than S_N for this problem.

While NN prediction was fast compared to S_N, Table 2 shows that NN training was four orders of magnitude slower than S_N, and one order of magnitude slower than MC. These timings indicate that, for the platform under test, NNs are only profitable if the cost of NN training may be amortized with multiple NN predictions.

4. CONCLUSIONS

While ANNs have shown great success in tasks such as regression and classification, the use of ANNs for solving differential equations is not as mature. As a result, though ANNs are capable of solving problems such as the NTE in a slab geometry, it comes with many more design decisions (e.g., number of nodes) and heuristics (e.g., using particular values for regularization) than simply

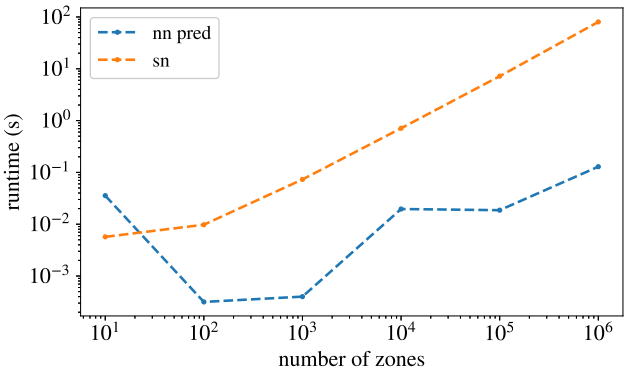

Figure 5: Runtime by zone count for Problem 1.

using traditional methods such as S_N or Monte Carlo. Nonetheless, using ANNs to approach problems such as the NTE is still an interesting avenue of research, and should be studied further.

In particular, some areas of interest include varying the ANN parameters (e.g., number of nodes in the hidden layer, etc.) to preserve symmetry in the solution for $\hat{\Phi}$ and accelerate convergence. Additionally, other measures of accuracy can be explored to help understand how the character of the ANN solution differs from S_N and MC. Furthermore, the ANN can be run on GPUs to achieve greater speedups. Finally, the ANN can be used to solve more complicated problems, for example, a problem including scattering or a non-uniform source (e.g., distributed over half of the slab).

ACKNOWLEDGEMENTS

The author would like to thank Kyle Bilton for his tireless work on this project, Jasmina Vujic for asking us to do a project, and Patrick Brantley for suggesting machine learning. This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-CONF-772639.

REFERENCES

[1] M. Raissi, P. Perdikaris, and G. E. Karniadakis. "Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations." Technical report, Brown University, University of Pennsylvania (2017).

[2] M. M. Chiaramonte and M. Kiener. "Solving differential equations using neural networks." http://cs229.stanford.edu/proj2013/ChiaramonteKiener-SolvingDifferentialEquationsUsingNeuralNetworks.pdf (2013). Accessed: 2018-12-08.

[3] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press (2016). http://www.deeplearningbook.org.

[4] K. Sato, C. Young, and D. Patterson. "An in-depth look at Googles first Tensor Processing Unit (TPU)." https://cloud.google.com/blog/products/gcp/

an-in-depth-look-at-googles-first-tensor-processing-unit-tpu (2017). Accessed: 2018-12-08.

[5] "Nvidia Tensor Cores." https://www.nvidia.com/en-us/data-center/tensorcore. Accessed: 2018-12-08.

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. "Scikit-learn: Machine Learning in Python." *Journal of Machine Learning Research*, **volume 12**, pp. 2825–2830 (2011).

[7] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems." (2015). URL https://www.tensorflow.org. Software available from tensorflow.org.

[8] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. "Automatic Differentiation in PyTorch." (2017).

[9] P. S. Brantley. "Spatial Treatment of the Slab-geometry Discrete Ordinates Equations Using Artificial Neural Networks." Technical Report UCRL-JC-143205, Lawrence Livermore National Laboratory, Livermore, California (2001).

[10] J. Vujic. "Derivation of Slab Geometry Neutron Transport Equation." Personal communication.

[11] E. Larsen. "Derivation of Slab Geometry Discrete Ordinates Equations." Personal communication.

[12] E. E. Lewis and J. Miller, W. F. *Computational methods of neutron transport*. Wiley (1984).

[13] D. P. Kingma and J. Ba. "Adam: A Method for Stochastic Optimization." *ArXiv e-prints* (2014).

[14] L. M. Leemis. *Probability*. Lightning Source (2011). http://www.math.wm.edu/~leemis.

[15] J. Vujic. "Derivation of Linearly-Anisotropic Sampling Algorithm." Personal communication.

[16] P. S. Brantley. "Derivation of Distance to Collision Sampling Algorithm." Personal communication.

[17] J. Vujic. "Derivation of Analytic Flux Solution." Personal communication.

[18] "Machine Learning for Neutron Transport Github Page." https://github.com/kjbilton/neutron_transport_net. Accessed: 2018-12-08.