Compact Neural Network Solutions to Laplace's Equation in a Nanofluidic Device

Martin Magill
U. of Ontario Inst. of Tech.
martin.magill1@uoit.net

Faisal Z. Qureshi
U. of Ontario Inst. of Tech.
faisal.qureshi@uoit.ca

Hendrick W. de Haan
U. of Ontario Inst. of Tech.
hendrick.dehaan@uoit.ca

Abstract

We explore the use of neural networks to solve the Laplace equation in a two-dimensional geometry. Specifically, we study a PDE problem that models the electric potential inside the slit-well nanofluidic device. Such devices are typically used to separate polymer mixtures by molecular size. Processes like these are commonly studied using GPU-accelerated coarse-grained particle simulations, for which GPU memory is a bottleneck. We compare the memory required to represent the field using neural networks to that needed to store solutions obtained using the finite element method. We find that even simple fully-connected neural networks can achieve accuracy to memory consumption ratios comparable to the good finite element solutions. These preliminary results demonstrate an industrial application that would benefit greatly from compact neural network representation techniques.

1 Introduction

Micro- and nanofluidic devices (MNFDs) consist of small geometries filled with electrolytic solution, such as water with dissolved NaCl [1–3]. One major application of these devices is the separation of polymer mixtures by size; specifically, the process of sorting DNA molecules by size is of widespread importance. MNFDs are being investigated as next-generation separation technologies for advantages such as miniaturization, automation, as well as improved speed and efficiency. Separation in MNFDs is often accomplished by introducing the polymers into the confined solution, then applying an electric field to drive them through the geometry. Over time, as a result, polymers become spatially segregated by size.

As the dynamics of polymers driven through confinement can be quite rich, MNFD design is research-intensive. Experimental investigations can be expensive, and have some intrinsic limitations (such as the optical resolution limit of light). Molecular dynamics (MD) simulations are often used in tandem with experiments, as they are cheaper and provide information that is inaccessible in experiment [4]. These simulations must balance physical realism against computational cost. Efficiency is particularly important because, as polymer dynamics in MNFDs are stochastic, simulations must be repeated many times to accurately measure statistical information. Simulations of MD in complete atomistic detail are quite computationally expensive. Coarse-grained (CG) models can be simulated far more efficiently, but must preserve sufficient detail to capture the essential phenomenology of the system under consideration. In coarse-grained Langevin dynamics (CGLD) models, a polymer is modelled as a chain of beads connected by springs, whereas in coarse-grained Brownian dynamics (CGBD) models an entire polymer is represented by a single effective particle. CGLD is often appropriate for nanofluidic devices, and CGBD for microfluidic devices.

These CG models can be simulated very efficiently, especially because they can readily exploit GPU acceleration. HOOMD-blue, a free open-source particle general purpose particle simulation toolkit, can accelerate simulations by over an order of magnitude using a single GPU, and achieves strong

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

scaling across up to thousands of GPUs [5, 6]. Furthermore, whereas published benchmarks of HOOMD-blue are conducted on simulations of millions of interacting particles, CG simulations for MNFDs often contain a few hundred particles or less. Since each simulation consumes so little memory, many independent instances can be simulated in parallel on the same GPU. This means nearly-perfect scaling is attainable in theory, limited only by available GPU memory.

One of the challenges of using GPUs to accelerate particle simulations of polymers driven through MNFDs is incorporating accurate electric field solutions. The standard methods for solving electric fields in complicated geometries (e.g. finite volume or finite element methods) are mesh-based. Users must decompose the problem domain into a mesh, which is often a time-consuming (and, therefore, expensive) process. The resulting field solution is represented in memory as a table containing several numbers per mesh point. Because field must be evaluated repeatedly during simulations, and since GPU-to-CPU communication is slow, it usually necessary for the field to be stored and evaluated in the GPU memory during simulations. Alas, the memory consumed by the mesh-based field solution directly reduces the CG simulation efficiency by displacing potential threads. Since mesh-based solution accuracy is proportional to mesh resolution, this memory cost cannot easily be reduced.

In this work, we use fully-connected tanh neural networks (NNs) to solve the electric potential in a MNFD geometry. We argue that NN solutions are particularly well-suited for GPU-accelerated CG particle simulations. First, they remove the need for custom mesh design. Second, as we showed in Magill et al. [7], NN representations of PDE solutions are overparametrized, suggesting the compact representation techniques could be used to significantly reduce their memory consumption. We solve the electric potential in the so-called slit-well device (Fig 1(a)) [3, 8–10]. We obtain a reference solution to the potential using the finite element method (FEM) with a high-resolution mesh. We compare the relative accuracy of NN solutions to FEM solutions as a function of memory consumption. The best NNs perform on par with the best FEM solutions, even without compact representation techniques.

2 Methodology

Dissanayake and Phan-Thien [11] showed that neural networks (NNs) can learn to approximate solutions to partial differential equation (PDE) problems using only the information available in the problem statements themselves. Many authors have since explored variations on this method (see [12–17] and others). Berg and Nyström [15] demonstrated that deeper NNs achieved better accuracy for a given memory cost. Sirignano and Spiliopoulos [16] and Han et al. [17] demonstrated that deep NNs could actually solve PDEs in hundreds of dimensions, which is a revolutionary feat. Whereas the computational cost (in memory and time) of mesh-based solvers grows exponentially in the dimensionality of the PDE, Grohs et al. [18] recently released a proof that the cost of solving Black-Scholes PDEs with deep NNs only grows at most polynomially in the dimensionality.

The electric potential in the slit-well device, u, can be modelled by the following PDE problem:

$$\nabla^2 u(x,y) = 0, \qquad (x,y) \in \Omega, \tag{1}$$

$$u(x,y) = 1, \qquad (x,y) \in (\partial\Omega)_1, \tag{2}$$

$$u(x,y) = -1, \qquad (x,y) \in (\partial\Omega)_2, \tag{3}$$

$$u_{\hat{n}}(x,y) = 0, \qquad (x,y) \in (\partial\Omega)_i, i \in \{3,4,\ldots,10\}, \tag{4}$$

where Ω is the problem domain, whose boundaries $(\partial\Omega)_i, i \in \{1,2,\ldots,10\}$ are illustrated in Fig 1(a). Fully-connected tanh NNs $\tilde{u}(x,y;\vec{p})$ were trained to minimize the loss function

$$\mathcal{L}[u] = \int_\Omega \|\nabla^2 u\|^2 dA + \lambda \int_{\partial\Omega} \|B[u]\|^2 dS, \tag{5}$$

where $B[u]$ encodes the boundary conditions (BCs) given by Eqns 2-4. As n minimizes $\mathcal{L}[n]$, it also approximates u. Training is accomplished by batch stochastic gradient descent. The training data are coordinates $\vec{x}_i$ randomly drawn from Ω; this is equivalent to numerically approximating $\mathcal{L}[\tilde{u}]$ by the Monte Carlo method. Because the boundaries have measure zero in Ω, we treat any training points within a small distance s from the boundaries as being on the nearest boundary. Given this approximation, our loss function effectively has a weighting factor of $\lambda = 10$. To differentiate training difficulties from representational limitations, we also trained NNs by supervised learning to mimic the reference FEM solution. We will refer to training on Eqns 1-4 as the indirect method, and training against the reference solution as the supervised method.

3 Results

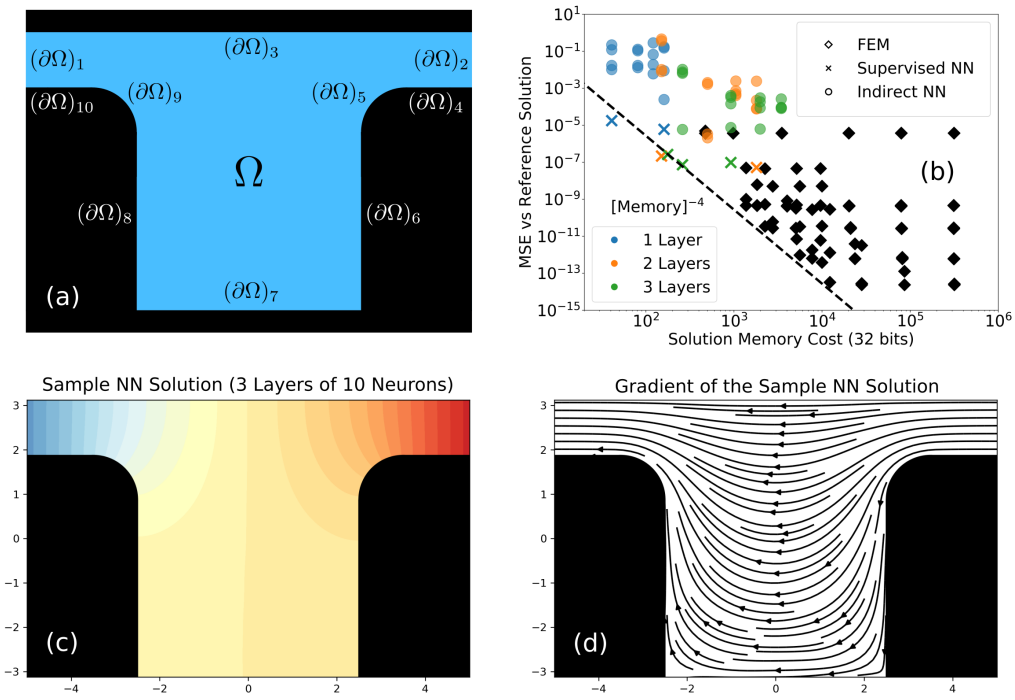

Figure 1: (a) Illustration of the PDE domain. The black boundaries are perfect insulators, modelled by homogeneous Neumann conditions. The two remaining boundaries carry uniform non-homogeneous Dirichlet conditions representing applied voltages. (b) The mean squared error of various numerical solutions relative to the high-resolution reference FEM solution, against the memory cost in increments of 32 bits. NNs trained using Eqns 1-4 are shown as circles, and NNs trained to mimic the reference solution are shown as crosses. Marker color indicates the depth of the networks. The black diamonds show FEM solutions. The dotted reference line scales as the -4th power of memory consumption. Panels (c) and (d) show a solution learned by an NN directly from Eqns 1-4.

Fig 1(b) compares the accuracy and memory costs (calculated as per App. A) of various NN and FEM solutions. As expected, the FEM solution converge to the reference FEM solution with increasing mesh density. Although the NN solutions exhibited worse accuracy, the best NN accuracies are still acceptable for most CG simulations. Fig 1(c) and (d) illustrate the NN solution corresponding to the bottom-left-most green circle marker in Fig 1(b), which is a good approximation to the true solution.

The dotted line in Fig 1(b) indicates the rough scaling of memory and accuracy observed amidst the best FEM solutions. Although the NNs trained from Eqns 1-4 do not attain this level of performance, some of the NNs trained in a supervised fashion do.

These results demonstrate the feasibility of using NNs to represent solutions to PDE problems for use in GPU-accelerated particle simulations. At the very least, the NN method has the appeal of removing the need for expensive mesh design. Future work will explore higher-dimensional problems, where the relative compactness of NN solutions over mesh-based solutions is expected to become far more pronounced. Indeed, whereas the memory cost of FEM solutions grows exponentially with increasing dimension, Grohs et al. [18] showed it grows polynomially for NNs. Three-dimensional MNFDs are common; time-varying fields in such devices produce four-dimensional fields. Furthermore, in many MNFDs, distortions of the electric field by molecular motion are important. Accounting for this coupling increases the dimensionality of the problem linearly in the number of moving particles. In the higher-dimensional models, the use of compact representation techniques will be crucial if the NN solutions are to be incorporated into GPU-accelerated simulations. In App. B, we have included estimates of the memory costs of FEM solutions for increasing problem dimension.

References

[1] Rafael Mulero, Anmiv S Prabhu, Kevin J Freedman, and Min Jun Kim. Nanopore-based devices for bioanalytical applications. *JALA: Journal of the Association for Laboratory Automation*, 15(3):243–252, 2010.

[2] Stephen L Levy and Harold G Craighead. DNA manipulation, sorting, and mapping in nanofluidic systems. *Chemical Society Reviews*, 39(3):1133–1152, 2010.

[3] Kevin D Dorfman. DNA electrophoresis in microfabricated devices. *Reviews of Modern Physics*, 82(4): 2903, 2010.

[4] Gary W Slater, Christian Holm, Mykyta V Chubynsky, Hendrick W de Haan, Antoine Dubé, Kai Grass, Owen A Hickey, Christine Kingsburry, David Sean, Tyler N Shendruk, and Lixin Zhan. Modeling the separation of macromolecules: A review of current computer simulation methods. *Electrophoresis*, 30(5): 792–818, 2009.

[5] Joshua A. Anderson, Chris D. Lorenz, and A. Travesset. General purpose molecular dynamics simulations fully implemented on graphics processing units. *Journal of Computational Physics*, 227 (10):5342 – 5359, 2008. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2008.01.047. URL `http://www.sciencedirect.com/science/article/pii/S0021999108000818`.

[6] Jens Glaser, Trung Dac Nguyen, Joshua A. Anderson, Pak Lui, Filippo Spiga, Jaime A. Millan, David C. Morse, and Sharon C. Glotzer. Strong scaling of general-purpose molecular dynamics simulations on gpus. *Computer Physics Communications*, 192:97 – 107, 2015. ISSN 0010-4655.

[7] Martin Magill, Faisal Qureshi, and Hendrick W de Haan. Neural Networks Trained to Solve Differential Equations Learn General Representations. *arXiv preprint arXiv:1807.00042*, 2018.

[8] Jongyoon Han and Harold G Craighead. Separation of long DNA molecules in a microfabricated entropic trap array. *Science*, 288(5468):1026–1029, 2000.

[9] Kuang-Ling Cheng, Yu-Jane Sheng, Shaoyi Jiang, and Heng-Kwong Tsao. Electrophoretic size separation of particles in a periodically constricted microchannel, 2008.

[10] Hanyang Wang, Gary Slater, and Hendrick Haan. Electrophoretic ratcheting of spherical particles in a simple microfluidic device: making particles move against the direction of the net electric field. In *APS March Meeting Abstracts*, 2017.

[11] M. W. M. G. Dissanayake and N. Phan-Thien. Neural-network-based approximations for solving partial differential equations. *Communications in Numerical Methods in Engineering*, 10(3):195–201, 1994. doi: 10.1002/cnm.1640100303. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/cnm.1640100303`.

[12] B. Ph. van Milligen, V. Tribaldos, and J. A. Jiménez. Neural Network Differential Equation and Plasma Equilibrium Solver. *Phys. Rev. Lett.*, 75:3594–3597, Nov 1995. doi: 10.1103/PhysRevLett.75.3594. URL `https://link.aps.org/doi/10.1103/PhysRevLett.75.3594`.

[13] Isaac E Lagaris, Aristidis Likas, and Dimitrios I Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, 9(5):987–1000, 1998.

[14] Neha Yadav, Anupam Yadav, and Manoj Kumar. *An introduction to neural network methods for differential equations*. Springer, 2015.

[15] Jens Berg and Kaj Nyström. A unified deep artificial neural network approach to partial differential equations in complex geometries. *Neurocomputing*, 317:28 – 41, 2018. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2018.06.056. URL `http://www.sciencedirect.com/science/article/pii/S092523121830794X`.

[16] Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339 – 1364, 2018. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2018.08.029.

[17] Jiequn Han, Arnulf Jentzen, and Weinan E. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34):8505–8510, 2018. ISSN 0027-8424. doi: 10.1073/pnas.1718942115. URL `http://www.pnas.org/content/115/34/8505`.

[18] Philipp Grohs, Fabian Hornung, Arnulf Jentzen, and Philippe von Wurstemberger. A proof that artificial neural networks overcome the curse of dimensionality in the numerical approximation of black-scholes partial differential equations. *arXiv preprint arXiv:1809.02362*, 2018.

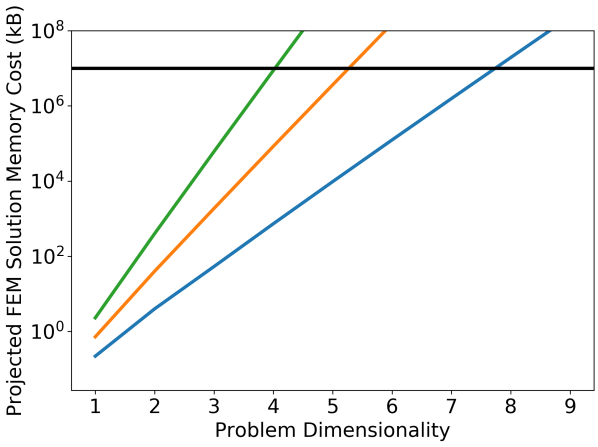

Figure 2: Estimated memory cost of FEM solutions with increasing problem dimensionality assuming constant mesh density. The horizontal black line indicates 10GB, which is the order of magnitude of memory available on most modern GPUs. The three coloured lines consume 4kB, 40kB, and 400kB of memory, respectively, when the dimensionality is 2.

A Calculating memory costs

The numer of parameters required to store a fully-connected NN is

$$3w + w(w+1)(d-1) + (w+1), \tag{6}$$

where w is the network's width and d is its depth. Each parameter was stored as single precision floating point numbers, so Eqn. 6 is the memory cost of an NN solution in increments of 32 bits.

The FEM solutions were stored in memory with 3 integers and 3 double precision floating point numbers at every mesh point. Allocating 16 bits per integer and 64 bits per double, the memory cost of a FEM solution in increments of 32 bits is thus

$$2(3N) + 0.5(3N) = 7.5N, \tag{7}$$

where N is the number of mesh points.

The values of w or N for the points in Fig. 1 are implied by Eqn. 6 and 7. The NNs had widths from 8 to 40, and the FEM meshes had roughly 50 to 40,000 mesh points.

B Estimating FEM memory costs in higher dimensions

For every increment that the problem dimensionality is increased, the FEM solution representations will require one extra integer and one extra double at every mesh point. In general, the memory cost of a FEM solution in increments of 32 bits is thus

$$2.5(D+1)N, \tag{8}$$

where N is the number of mesh points and D is the dimensionality of the problem.

Figure 2 estimates how the memory cost of FEM solutions would scale with problem dimensionality according to Eqn. 8 if the average mesh density is kept constant. The green line corresponds roughly to the densest FEM solutions in Fig. 1; this density would saturate a standard GPU's memory in a four-dimensional domain. The blue line corresponds roughly to the sparsest FEM solutions in Fig. 1; even this density saturates a standard GPU's memory in only eight dimensions.

For reference, modelling the particle-field coupling for a rather small polymer consisting of only 10 monomers in the two-dimensional slit-well device would produce a 22-dimensional PDE. Sirignano and Spiliopoulos [16] demonstrated that NNs could solve such problems with relative ease. Nonetheless, compact representation techniques will almost certainly be necessary to ensure that the NN solutions can be incorporated efficiently into GPU-accelerated simulations. Specifically, the results of Magill et al. [7] suggest that the NN solutions to PDEs are likely to be vastly overparametrized (as is generally the case in other NN applications as well).