Investigation of Weight Reuse in Multi-Layer Perceptron Networks for Accelerating the Solution of Differential Equations

Kevin McFall
Department of Mechanical Engineering
Georgia Institute of Technology – Lorraine Campus
2-3 Rue Marconi, 57070 Metz
France
kevin.mcfall@georgiatech-metz.fr

J. Robert Mahan
Department of Mechanical Engineering
Georgia Institute of Technology – Lorraine Campus
2-3 Rue Marconi, 57070 Metz
France
robert.mahan@georgiatech-metz.fr

***Abstract* – Research has shown that training multi-layer perceptron networks to solve ordinary and partial differential equations (DEs) can be accelerated by reusing network weights from a previously solved similar problem. This paper compares weight reuse for two existing methods of defining the network error function. Weight reuse is shown to accelerate training of one ordinary and two partial DEs even for equations with significantly different parameters or boundary/initial conditions. The second method outperforms the first for partial DEs where multiple boundary/initial conditions are defined, but fails unpredictably when weight reuse is applied to accelerate solution of the diffusion equation.**

I. INTRODUCTION

Artificial neural networks (ANN) provide an effective tool in solving a large variety of differential equations (DE) [1-7]. Solution using ANNs offers several advantages over standard numerical methods. First, a continuous solution is obtained over the entire domain rather than simply at discrete points. Additionally, computational complexity does not increase considerably for higher dimensional problems since the number of parameters to be optimized remains constant. Also, round-off error propagation is not an issue in neural network solutions as it is with standard numerical methods. Finally, this approach handles boundary and/or initial conditions of any type. Since boundary and initial conditions are mathematically equivalent, the distinction between them is dropped in the remainder of this paper.

Solving differential equations requires optimization of the ANN with the additional difficulty of satisfying the boundary conditions. This added constraint can be dealt with in several ways. The error function to be minimized can be defined as a sum of two components: one concerned with satisfying the DE and another associated with the boundary conditions. This method can lead to competition between the two error components, and thus slow down the training process. More sophisticated methods use evolutionary algorithms to adjust weights [5], or reduce the constrained optimization task to an unconstrained one [6]. Also, coding the boundary conditions with radial basis functions [7] is advantageous for dealing with irregular boundaries [6].

Research in ANN methods for solving DEs has concentrated on solving specific, individual problems. However, DEs are parametric by nature; solutions to similar DEs have the same form albeit with differing spectral content, time constants, amplitudes, etc. Intuitively, the solution to one DE is expected to be helpful in solving another with slightly different parameters. Training an ANN to solve a second problem can begin with the final weights obtained after solving a first problem. This method has proven beneficial for similar problems [8] and has even been applied successfully to the solution of DEs [9].

The authors in [9] extended the idea to a class of problems: the problem class consisted of solutions to the same second order, ordinary DE where one of the boundary conditions changed. A single network was selected using evolutionary algorithms which could quickly solve the DE for all of the boundary conditions in the problem class [10]. Significant improvements were observed in both speed and accuracy in solving the DEs.

Results have proven that weight reuse increases both speed of training and overall accuracy if the problems in questions are sufficiently similar. The work in [10] already assumed, and correctly so, that DEs with a slightly modified boundary condition belong to the same problem class. The next obvious question is how similar must the DEs be in order for them to still belong to the same problem class? With an answer to this question, a system could be designed which quickly solves a large variety of differential equations. This system would first decide to which of the known problem classes a new DE belongs, and then solve it with the appropriate network for that problem class.

This paper launches an investigation into where the boundaries between problem classes in DEs lie. The problems are solved using multi-layer perceptron networks (MLPN). The benefits of weight reuse are examined for three DEs: the classic second-order, ordinary differential equation, the diffusion equation, and the potential equation. As mentioned previously, various methods exist for simultaneously satisfying both the DE and the boundary conditions. Two methods which make use of gradient descent optimization are employed, and their applicability to weight reuse is examined.

II. METHODS FOR ERROR FUNCTION DEFINITION

Consider the DE to be solved given by

$$G\left(\mathbf{x}, \psi(\mathbf{x}), \nabla\psi(\mathbf{x}), \nabla^2\psi(\mathbf{x}), \ldots\right) = 0; \ \mathbf{x} \in \Re^n, \tag{1}$$

where $\psi(\mathbf{x})$ denotes the solution subject to certain boundary conditions. The first method (hereafter referred to as method 1) for solving the equation defines the approximate solution as

$$\psi_t(\mathbf{x}) = N(\mathbf{x}, \boldsymbol{\theta}), \tag{2}$$

where N is the output of the MLPN and $\boldsymbol{\theta}$ is the vector of network weights to be optimized. The error function to be minimized is then

$$E = E_{\mathrm{DE}} + \eta E_{\mathrm{BC}}, \tag{3}$$

where the first term

$$E_{\mathrm{DE}} = \frac{1}{|D|}\sum_{\mathbf{x}_i \in D}\left[G\left(\mathbf{x}, \psi_t(\mathbf{x}), \nabla\psi_t(\mathbf{x}), \nabla^2\psi_t(\mathbf{x}), \ldots\right)\right]^2 \tag{4}$$

accounts for error in the differential equation itself and the second term

$$E_{\mathrm{BC}} = \frac{1}{|S_1|}\sum_{\mathbf{x}_i \in S_1}\left[\psi_t(\mathbf{x}_i) - \psi_{\mathrm{BC}}(\mathbf{x}_i)\right]^2 + \frac{1}{|S_2|}\sum_{j=1}^{|\mathbf{x}|}\sum_{x_i^{(j)} \in S_2}\left[\left.\frac{\partial\psi_t}{\partial x^{(j)}}\right|_{x_i^{(j)}} - \left.\frac{\partial\psi_{\mathrm{BC}}}{\partial x^{(j)}}\right|_{x_i^{(j)}}\right]^2 \tag{5}$$

accounts for error in satisfying the boundary conditions with a weighting factor of $\eta = 10$ as recommended by [9]. Equations (4) and (5) are defined where D is given by a finite set of points within the desired domain, S_1 is the set of points where the boundary value ψ_{BC} is specified, S_2 is the set of points where the boundary derivate $\partial\psi_{\mathrm{BC}}/\partial x^{(j)}$ is specified, and $x^{(j)}$ is the j^{th} component in the vector $\mathbf{x}$. Error defined in (3) is the same method used in [9-10] with the added possibility of Neuman boundary conditions.

The second method (hereafter denoted method 2) for defining the error function involves recasting the approximate solution to the differential equation so that the boundary conditions are automatically satisfied. The approximate solution takes the form

$$\psi_t(\mathbf{x}) = A(\mathbf{x}) + F(\mathbf{x}, N(\mathbf{x}, \boldsymbol{\theta})). \tag{6}$$

The function A satisfies the boundary conditions, while F is chosen to return zero at all of the boundaries. A systematic approach exists for finding A and F for any Dirichlet and/or Neuman conditions on a uniform boundary [6]. Since the boundary conditions are automatically satisfied, the error function

$$E = E_{\mathrm{DE}} \tag{7}$$

is sufficient to solve the problem. As an example, consider the classic second-order ordinary DE (hereafter called the oscillator equation)

$$m\ddot{x} + R\dot{x} + kx = 0 \tag{8}$$

with initial conditions

$$x(0) = a \text{ and } \left.\frac{dx}{dt}\right|_{t=0} = b. \tag{9}$$

The approximate solution to (8) and (9) would then be

$$x_t = a + t\left[b + tN(t, \boldsymbol{\theta})\right], \tag{10}$$

which automatically satisfies the initial conditions no matter what the output of the MLPN.

Minimizing the error functions in either (3) or (7) involves the various partial derivatives of N with respect to the inputs $x^{(j)}$ and the weights in the $\boldsymbol{\theta}$ vector. Determination of these partial derivatives is detailed in [6] for an MLPN structure with one hidden layer of nodes with logarithmic sigmoid transfer functions and a linear output node without a bias. The partial derivative of error with respect to the weights, $\partial E/\partial\boldsymbol{\theta}$, is then applied using the RPROP algorithm [11] to update the network weights. All of the MLPNs used contain a hidden layer of 10 nodes.

III. SOLVING THE OSCILLATOR EQUATION

Consider Fig. 1 illustrating the solutions to the ordinary DE in (8) with initial conditions (9) for various values of m, R, and k with $a = 1$, and $b = 0$. The solid curve **1** in Fig. 1 is under-damped and will be used as the base problem to be learned. The other three curves include under-damped, over-damped, and critically-damped solutions which will be used to produce the weights for reuse.

Results for both methods 1 and 2 appear in Fig. 2 for solving problem **1** when reusing the weights after first solving problems **2** through **4**. The curves in Fig. 2 plot the RMS error of the approximate solution as a function of training epoch averaged over 25 runs. The error functions (3) and (7) for methods 1 and 2 respectively are evaluated on a domain of 10 equally spaced points in time.

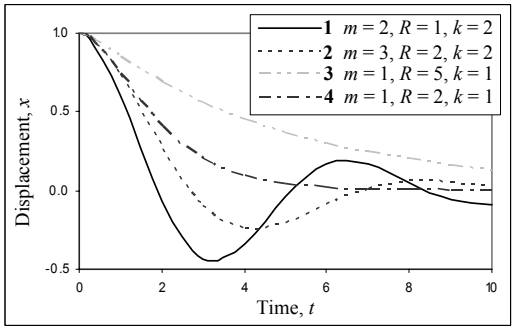


Fig. 1. Four solutions to the DE defined by (6) and (7) with $a = 1$ and $b = 0$.

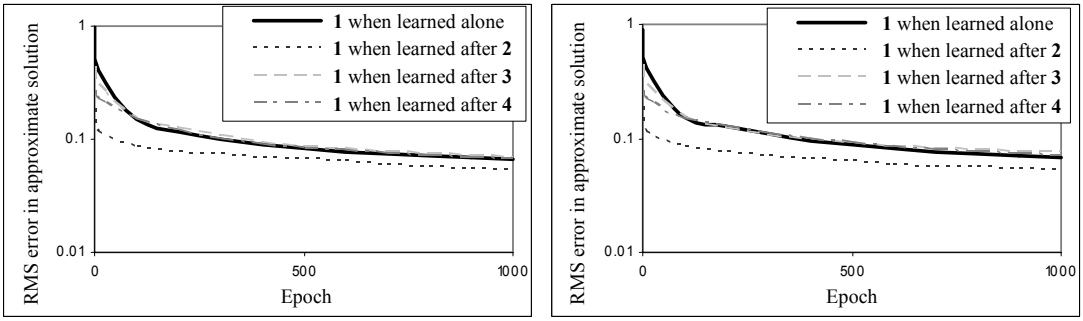


Fig. 2. Learning curves for solving the oscillator problem **1** after reusing weights from the other three problems for both method 1 (left) and method 2 (right).

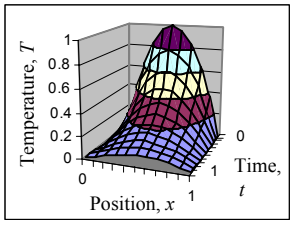


Fig. 3. Solution of the diffusion equation for problem **1**.

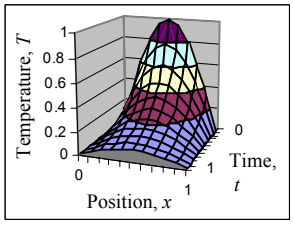


Fig. 4. Solution of the diffusion equation for problem **2**.

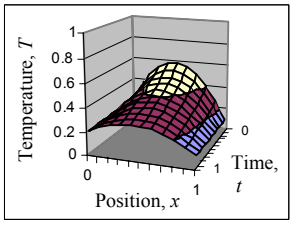


Fig. 5. Solution of the diffusion equation for problem **3**.

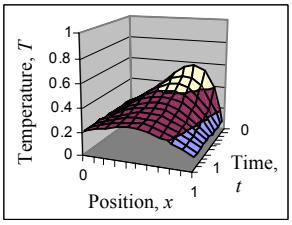
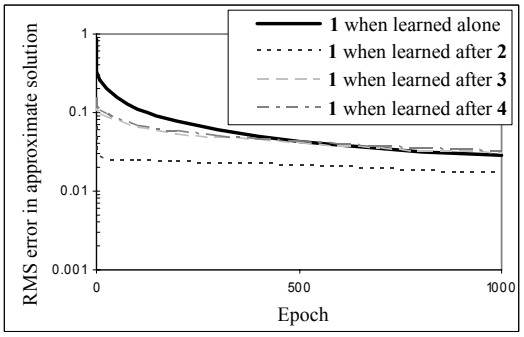


Fig. 6. Solution of the diffusion equation for problem **4**.



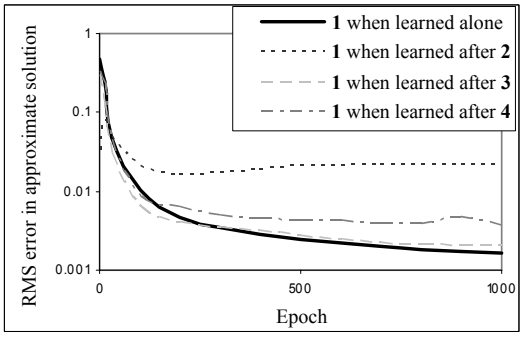


Fig. 7. Learning curves for solving the diffusion equation problem **1** after reusing weights from the other three problems for both method 1 (left) and method 2 (right).

Notice first that the training curves for both methods are essentially identical; neither method provides an advantage over the other. Automatic satisfaction of the single initial condition at $t = 0$ apparently provides little advantage when using method 2. Also, significant improvement with weight reuse is only observed when first trained with problem **2**, which is the other under-damped case. However, the improvement is not on the order of a factor of ten as reported in [10]. This reduction in performance is expected as the entire shape of the solution curve is different, whereas [10] investigated only slight changes in a single boundary condition or single parameter of the differential equation. It is also not surprising that improvement is only observed for the case of problem **2**, whose solution is certainly closest to the solution to problem **1**.

These results indicate that weight reuse does accelerate training, at least for the case of similar damping. More cases must be studied, including varying the initial condition as well, to discover how far-reaching this conclusion might be.

IV. SOLVING THE HEAT DIFFUSION EQUATION

The second problem investigated is the heat diffusion equation given by

$$\alpha \frac{\partial^2 T(x,t)}{\partial x^2} - \frac{\partial T(x,t)}{\partial t} = 0 \tag{11}$$

with boundary conditions

$$T(0,t) = a\ ,\ T(L,t) = b \ \text{ and } \ T(x,0) = f(x). \tag{12}$$

Four sets of boundary conditions are examined for this DE as well. Problem **1**, whose solution is plotted in Fig. 3, is the base problem to be solved with an initial condition of

$$f(x) = T_{\max} \sin \frac{\pi x}{L} \tag{13}$$

with a peak at $T_{\max} = 1$ and homogenous boundary conditions $a = b = 0$. This DE leads to a simple analytic solution in order to evaluate the fitness of the approximate solution. Problems **2** through **4** are used to generate the weights for reuse, and their solutions appear in Figs. 4 through 6. Problem **2** also has homogenous boundary conditions $a = b = 0$ while problems **3** and **4** have $a = 0.2$ and $b = 0.1$. Problems **2** and **3** have parabolic initial conditions

$$f(x) = c_2 x^2 + c_1 x + c_0 \tag{14}$$

where coefficients c_0, c_1, and c_2 are chosen to satisfy the boundary conditions as well producing peaks at $T_{\max} = 1$ and $T_{\max} = \frac{3}{5}$ for problems **2** and **3** respectively. Problem **4** has the quartic initial condition

$$f(x) = c_4 x^4 + c_2 x^2 + c_0 \tag{15}$$

where coefficients c_0, c_2, and c_4 are again chosen to satisfy the boundary conditions and produce a peak of $T_{\max} = \frac{3}{5}$. Notice that the peaks in for problems **3** and **4** are not centered in space as they are for problems **1** and **2**.

The training curves for weight reuse in the diffusion equation appear in Fig. 7, where results are again averaged over 25 runs and the error function is evaluated on a 10×10 grid of points. The solutions to problems **1** and **2** are so similar that one certainly expects an improvement when reusing weights between these two problems. This expectation is confirmed for method 1, but results for method 2 are catastrophically poor. The cause for this failure is unknown. Results improve somewhat when the MLPN is trained with a more dense 15×15 grid, but even

these results are still worse than when learning problem **1** alone without weight reuse. The standard deviation of the error over the 25 runs is much larger for this DE than either of the other two examined; most runs produce a reasonably small error while a few isolated runs destroy the average with errors orders of magnitude larger. More investigation is necessary to uncover why experimental observations so clearly contradict intuition for this case – and only for method 2.

Unlike solution of the oscillator equation, method 2 outperforms method 1 significantly. After only 200 epochs, method 2 has reached an error nearly an order of magnitude lower than method 1 after 2000 epochs. The reason for the difference is that the approximate solution

$$T_t = f(x) + xt(x-L)N(x,t,\boldsymbol{\theta}) \tag{16}$$

automatically satisfies the DE at three of the four boundaries. This is apparent in Fig. 8, which illustrates the approximate solutions for both methods after 250 epochs of training. The correct solution is only beginning to take shape for method 1, whereas values between the boundaries need simply be "filled in" between the three given boundaries for method 2. Even if method 1 offers significantly accelerated training with weight reuse, results are still better using method 2 without weight reuse.

V. SOLVING THE POTENTIAL EQUATION

The final problem to be investigated is the potential equation given by

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \tag{17}$$

with boundary conditions

$$u(0,y) = f_0(y),\ u(1,y) = f_1(y), \qquad u(x,0) = g_0(x),\ \text{and}\ u(x,1) = g_1(x). \tag{18}$$

Solutions to the four problems used appear in Figs. 9 through 12, where problem **1** is again used as the base and the other three for weight reuse. Problem **1** has a simple analytic solution with boundary conditions

$$g_1(x) = \sin \pi x \tag{19}$$

and

$$g_0(x) = f_0(y) = f_1(y) = 0. \tag{20}$$

All four problems share the boundary condition (19) but differ for the other three. These boundary conditions are

$$g_0(x) = \tfrac{1}{2},\ f_0(y) = f_1(y) = \tfrac{1}{2}(1-y), \tag{21}$$

$$g_0(x) = \tfrac{1}{2} - \tfrac{3}{10}(1-x),\ f_0(y) = \tfrac{1}{5}(1-y),\ f_1(y) = \tfrac{1}{2}(1-y),\ \text{and} \tag{22}$$

$$g_0(x) = 0 \text{ and } f_0(y) = f_1(y) = \tfrac{1}{2}\sin \pi y \tag{23}$$

for problems **2**, **3**, and **4** respectively.

The learning curves in Fig. 13 have essentially the same shape for both methods; weight reuse starting from problems **2** and **3** experiences significantly accelerated training while starting with problem **4** produces only

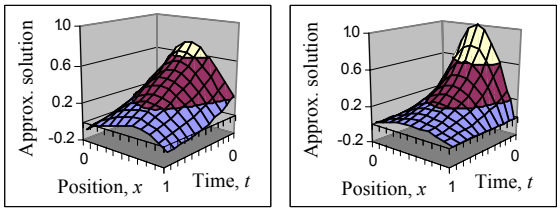


Fig. 8. Approximate solutions of the diffusion equation problem **1** after 250 epochs when using method 1 (left) and method 2 (right).

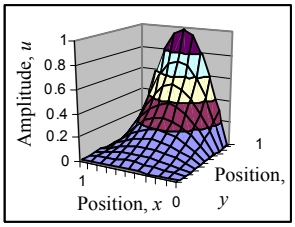


Fig. 9. Solution of the potential equation for problem **1**.

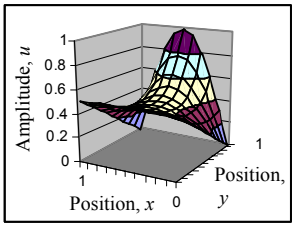


Fig. 10. Solution of the potential equation for problem **2**.

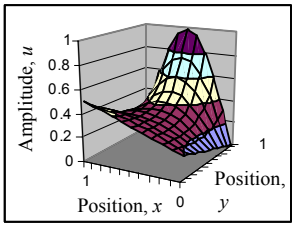


Fig. 11. Solution of the potential equation for problem **3**.

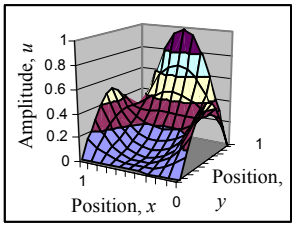
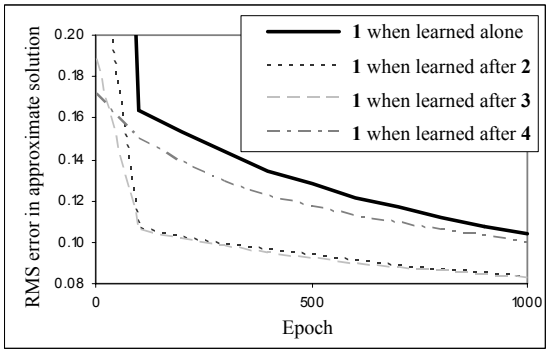


Fig. 12. Solution of the potential equation for problem **4**.



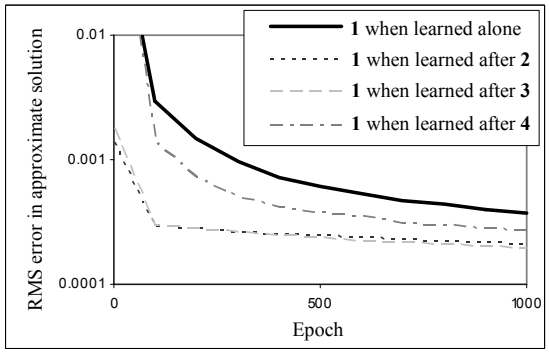


Fig. 13. Learning curves for solving the potential equation problem **1** after reusing weights from the other three problems for both method 1 (left) and method 2 (right). Note that *y*-axis scales are different for the two plots.

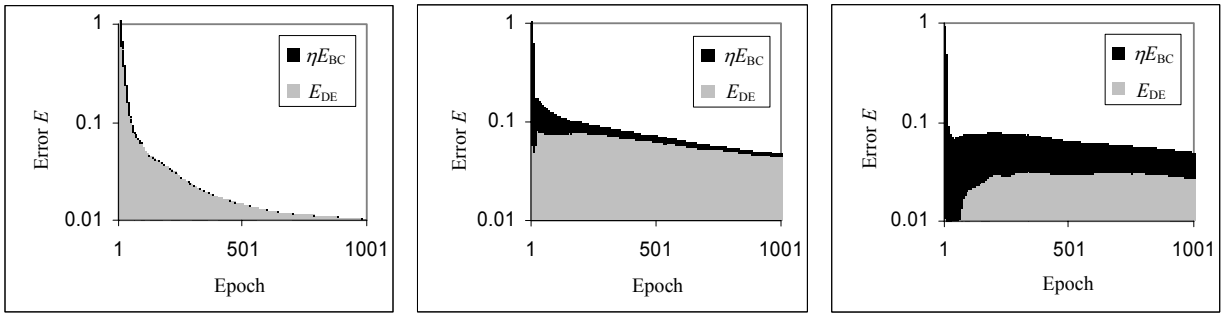


Fig. 14. Separation of training error into DE and boundary condition components for method 1 when solving problem **1** of the oscillator (left), diffusion (center), and potential (right) equations.

moderate improvement. The linear boundary conditions in problems **2** and **3** are evidently sufficiently similar to those in problem **1** to ensure improvement from weight reuse.

While the shapes of the learning curves are similar, the difference in absolute performance between methods 1 and 2 is dramatic (note the difference in vertical axis scales in Fig. 13). Adding the fixed boundary on the fourth side completely encloses the domain with an automatically satisfied boundary for method 2. Apparently the more boundaries which are specified, the faster and more accurate method 2 becomes.

Since the error function (3) for method 1 is composed of components satisfying both the DE and the boundary conditions, reducing the overall error involves a trade-off between competing components. Fig. 14 illustrates the breakdown of total error into these two components during training of problem **1** for the three DEs studied. The oscillator equation has a single initial condition and thus E_{BC} is easily reduced early during training – focusing the majority of training effort on satisfying the DE. For this reason, methods 1 and 2 produce similar results for the oscillator equation. Increasingly more effort must be channelled toward satisfying boundary conditions with method 1 when solving the diffusion and potential equations. This is especially apparent in Fig. 14 for the potential equation, as E_{BC} dominates the error in early training and overall error, E, in fact increases temporarily as the MLPN attempts to reduce it. As the constraints on the boundary conditions increase, interplay between reducing E_{BC} and E_{DE} impedes reduction of the overall error. Thus method 2 performs significantly better than method 1 for the diffusion and potential equations.

VI. CONCLUSION

Previous work has shown that multi-layer perceptron networks (MLPN) are powerful tools for solving various differential equations (DEs). Additionally, solution of a second DE with slightly different boundary conditions or parameters can be accelerated by reusing the weights from solving a previous problem. The intent of this research is to identify significantly different DEs that are still sufficiently similar to accelerate training with weight reuse. Results have revealed several cases where this is true.

Numerical experiments were conducted for two methods of defining the MLPN error function. The first (method 1) simply optimizes the MLPN so that its output is the desired approximate solution. The second method (method 2) rewrites the approximate solution so that the boundary conditions are automatically met, regardless of the MLPN output. Comparison of these two methods led to the intriguing observation that method 2 produces extremely accurate solutions in a small number of training epochs even without weight reuse. This second method is especially useful when solving partial DEs where several of the boundaries are specified, as in the cases of the diffusion and potential equations examined here.

Weight reuse with method 1 produced results which were largely expected: the more similar the second problem was to the first, the more weight reuse accelerates training. For example, training time to solve an under-damped problem (problem **1** in Fig. 1) to the same error was halved when starting from a different under-damped problem (problem **2** from Fig. 1). Starting from an over- or critically-damped problem (problems **3** and **4** respectively from Fig. 1) produced no appreciable acceleration however. Results for method 2 with weight reuse were similar for the oscillator and potential equations, but contradicted expectation for the diffusion equation. In this case, weight reuse in fact affected training detrimentally (see Fig. 7), and most significantly when the problems had nearly identical solutions (see Figs. 3 and 4). Rewriting the approximate solution to automatically satisfy boundary conditions apparently adds an extra dynamic which can cause weight reuse to fail completely.

This paper has shown that DEs with significantly different parameter values or boundary conditions are still sufficiently similar to experience accelerated training from weight reuse. What constitutes "similar" can be predicted intuitively when using method 1, at least for the DEs studied here. Method 2 generates lower error solutions than method 1 when multiple boundaries are specified, but does not always perform as expected when reusing weights (see solution of the diffusion equation). These results indicate the need for continued investigation in order to quantitatively classify DEs as similar with respect to weight reuse, and to understand and predict the unexpected behavior when using method 2.

VII. ACKNOWLEDGEMENT

The authors owe a debt of gratitude to the Conseil Régional de Lorraine for its generous financial support of research at the European Platform of the Georgia Institute of Technology located in Metz, France.

VIII. REFERENCES

[l] H. Lee, I. Kang, "Neural Algorithms for Solving Differential Equations", *Journal of Computational Physics*, vol. 91, pg. 110-117, 1990.

[2] A. Meade Jr., A. Fernandez, "The Numerical Solution of Linear Ordinary Differential Equations by Feedforward Neural Networks", *Mathematical Computational Modelling*, vol. 19, no. 12, pg. 1-25, 1994.

[3] R. Yentis, M. Zaghoul, "VLSI Implementation of Locally Connected Neural Network for Solving Partial Differential Equations", *IEEE Transactions on Circuits and Systems I*, vol. 43, no. 8 , pg. 687-690, 1996.

[4] D. Parisi, M. Mariani, M. Laborde, "Solving Differential Equations with Unsupervised Neural Networks", *Chemical Engineering and Processing*, vol. 42, issues 8-9, pg. 715-721, Aug.-Sept. 2003.

[5] L. Aarts, P. Van der Veer, "Neural Network Method for Solving Partial Differential Equations", *Neural Processing Letters*, vol. 14, pg. 261-271, 2001.

[6] I. Lagaris, A. Likas, D. Fotiadis, "Artificial Neural Networks for Solving Ordinary and Partial Differential Equations", *IEEE Transactions on Neural Networks*, vol. 9, no. 5, pg. 987-1000, Sept. 1998.

[7] L. Jianyu, L. Siwei, Q. Yingjian, H. Yaping, "Numerical Solution of Elliptic Partial Differential Equation by Growing Radial Basis Function Neural Networks", *Proceedings of the 2002 International Joint Conference on Neural Networks*, vol. 1, May 2002.

[8] S. Raudy, "Prior weights in adaptive pattern classification", *Proceedings from the 15th International Conference on Pattern Recognition*, volume 2, pg. 1010-1013, 2000.

[9] M. Hüsken, C. Goerick, "Fast learning for problem classes using knowledge based network initialization", *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks*, vol. 6, pg. 619-624, 25-27 July 2000.

[10] M. Hüsken, C. Goerick, A. Vogel, "Fast Adaptation of the Solution of Differential Equations to Changing Constraints", *Proceedings of the 2nd International ICSC Symposium on Neural Computation*, pg. 181-187, 2000.

[11] M. Riedmiller, H. Braun, "A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm", *IEEE International Conference on Neural Networks*, vol. 1, pg. 586-591, April 1993.