Draft version December 19, 2019
Preprint typeset using LaTeX style emulateapj v. 12/16/11

SOLVING A PARTIAL DIFFERENTIAL EQUATION AND AN EIGENVALUE PROBLEM WITH FEED-FORWARD NEURAL NETS

Sigbjørn Foss

Draft version December 19, 2019

ABSTRACT

We implement neural networks to solve a second order partial differential equation and an eigenvalue problem for a 6×6 symmetric positive-definite matrix. The neural net approximates the solution to the PDE decently, although not to the accuracy of preexisting numerical methods. The network for the eigenvalue problem finds the largest and smallest eigenvalues of all matrices that were tested with high accuracy. this link.

1. INTRODUCTION

'Out of the box' methods for solving partial differential equations and eigenvalue problems are highly sought after in many fields of science. Neural networks have been proposed as flexible methods for solving a wide variety of such problems, and the exploration of neural network methods is an active field of research. This paper applies neural networks to solve the heat flow PDE and to find the largest and smallest eigenvalues of a symmetric matrix. The results are compared to traditional methods. The methods section contains an overview of the background of the neural network methods that are applied to these problems, as well as the analytical and numerical methods traditionally applied to PDEs. In the results section, the results are presented and the conclusion compares and critiques the methods.

2. METHODS

2.1. *The Heat Flow Equation*

2.1.1. *Analytical Solution*

The one dimensional heat flow equation is a partial differential equation with two variables,

$$\frac{\partial^2 u(x,t)}{\partial x^2} = \frac{\partial u(x,t)}{\partial t}.$$

Assuming a separable solution

$$u = X(x)T(t), \tag{1}$$

both sides of the equation must equal a constant,

$$\frac{1}{X}\frac{\partial^2 X}{\partial x^2} = \frac{1}{T}\frac{\partial T}{\partial t} = \alpha$$

. Applying separation of variables, the time dependent equation has the solution

$$T(t) = Ce^{\alpha t}.$$

Inserting into (1) and using the condition that $u(x,0) = \sin \pi x$, the solution to the space dependent equation is found,

$$\begin{aligned} u(x,t) &= Ce^{\alpha t}X(x), \\ u(x,0) &= CX(x), \\ &= \sin \pi x \\ \Rightarrow X(x) &= \frac{1}{C}\sin \pi x. \end{aligned}$$

The equation holds for $\alpha = -\pi^2$, so the solution is

$$u(x,t) = e^{-\pi^2 x}\sin \pi x, \tag{2}$$

which also satisfies the remaining initial condition $u(0,t) = u(1,t) = 0$.

2.1.2. *Numerical Solution*

The heat flow equation can also be solved by the explicit forward Euler algorithm,

$$u_{xx} = u_{tt},$$

where

$$u_{xx} = \frac{u(x_i + \Delta t, t_j) - 2u(x_i, t_j) + u(x_i - \Delta x, t_j)}{\Delta x^2},$$

and

$$u_{tt} = \frac{u(x_i, t_j + \Delta t) - u(x_i, t_j)}{\Delta t}.$$

Let $u(x_i, t_j) = u_{ij}$. The equation is then

$$u_{i,i+1} = u_{ij} + \frac{\Delta t}{\Delta x^2}\left(u_{i+1,j} - 2u_{ij} + u_{i-1,j}\right).$$

This can be expressed by a matrix product. Let $u_{*,j} = \boldsymbol{u}_j$ and $\boldsymbol{A} \in \mathbb{R}^{N_x \times N_x}$ be a banded matrix with -2 on the diagonal and ones on the first off diagonals, then

$$\boldsymbol{u}_{j+1} = \boldsymbol{u}_j + \frac{\Delta t}{\Delta x^2}\boldsymbol{A}\boldsymbol{u}_j,$$

with $\boldsymbol{u}_0 = \sin \pi \boldsymbol{x}$.

2.1.3. *Neural network solution*

The heat flow equation can be written

$$\begin{aligned} \frac{\partial^2 u(x,t)}{\partial x^2} - \frac{\partial u(x,t)}{\partial t} &= 0 \\ f(u(x,t)) &= 0. \end{aligned}$$

A function u that minimizes f is then a good approximation to the solution of the differential equation. The idea is then to choose some trial function g which takes in x, t and the network output N, and define a cost function that depends on g. The trial solution should be defined so that it satisfies the initial conditions. This is achieved by letting

$$g(x,t,N) = h_1(x,t) + h_2(x,t,N),$$

where h_1 satisfies the initial conditions and h_2 is zero at he points where the initial conditions are evaluated, ensuring that the network output does not break the initial conditions. The initial conditions are in this case

$$h_1(x,0) = \sin \pi x,$$
$$h_1(0,t) = 0,$$
$$h_1(1,t) = 0.$$

Choosing

$$h_1(x,t) = (1-t)\sin \pi x$$

fulfils the conditions and

$$h_2(x,t,N) = x(1-x)tN,$$

evaluates to zero at $t = 0$, $x = 0$ and $x = 1$. Using the mean squared error $(f(g(x,t,N)) - 0)^2$ as the cost function, the cost function can be minimized using a neural network with gradient descent. The only significant difference from a regular classification task is that the derivatives in the cost functions must be evaluated. This is easy to implement, however, due to the automatic numerical differentiation tools in Tensorflow, or other libraries. The network then takes in a point (x, y), $x, y \in (0, 1)$ and outputs a value N, which is in turn used in the trial function to make a prediction of the function value at this point. The network is set up with, with ELU activation function in the hidden layers and using Nesterov accelerated gradient optimization. The discussion in this section is in large parts taken from Heins text [1].

2.2. *approximation of eigenvalues using neural networks*

Approximating eigenvalues of a matrix essentially amounts to a least square problem. A matrix A with an eigenvector v and corresponding eigenvalue λ fulfils

$$\boldsymbol{A}\boldsymbol{v} = \lambda \boldsymbol{v},$$
$$\boldsymbol{v}^T\boldsymbol{A}\boldsymbol{v} = \lambda \boldsymbol{v}^t\boldsymbol{v},$$
$$\lambda = \frac{\boldsymbol{v}^T\boldsymbol{A}\boldsymbol{v}}{\boldsymbol{v}^T\boldsymbol{v}}.$$

This expression for the eigenvalue is the Rayleigh quotient. An eigenvalue-eigenvector pair that minimizes the quotient corresponds to the smallest eigenvalue of A. Using $-A$ instead produces an approximation to the largest eigenvalue. Solving this with a neural network is then just a matter of using the Rayleigh coefficient as a cost function, with v as the network output. After training, the network output should be the eigenvector, from which the eigenvalue can be calculated. The network structure is somewhat different for this method compared to the PDE net. Here, since the input vectors are not drawn from a sample, instead, the output is fed into the network for each iteration, starting with a random input vector.

3. RESULTS

3.1. *Partial differential equation*

After a fair amount of testing, the network is chosen to have three hidden layers. Trying to minimize the cost function with other architectures and a variety of layer sizes, the network converges to give about the same error, having restricted the parameter space to 3 or less layers of less than a hundred neurons each, and the maximum number of epochs to $3 \cdot 10^5$. The main difference is that the three layer network converges faster in general. The same is true for choice of activation functions and optimization algorithm. Using ELU activation functions and Nesterov accelerated gradient descent makes the network converge faster, but the learning curve flattens and does not decrease the cost function below values in $[10^{-5}, 10^{-4}]$ (figure 1).

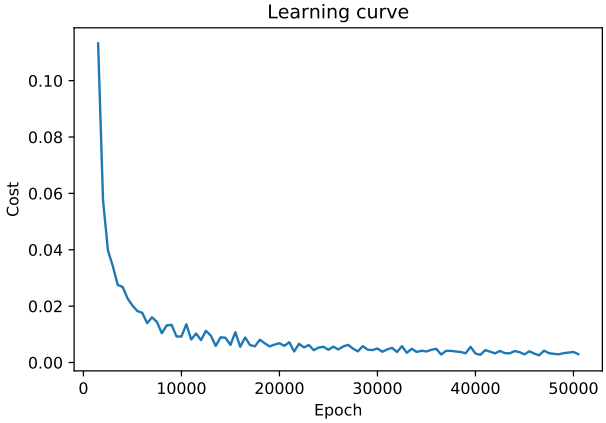

FIG. 1.— Learning curve for 50 000 iterations

The result from the three layer neural net is shown in figure 2.

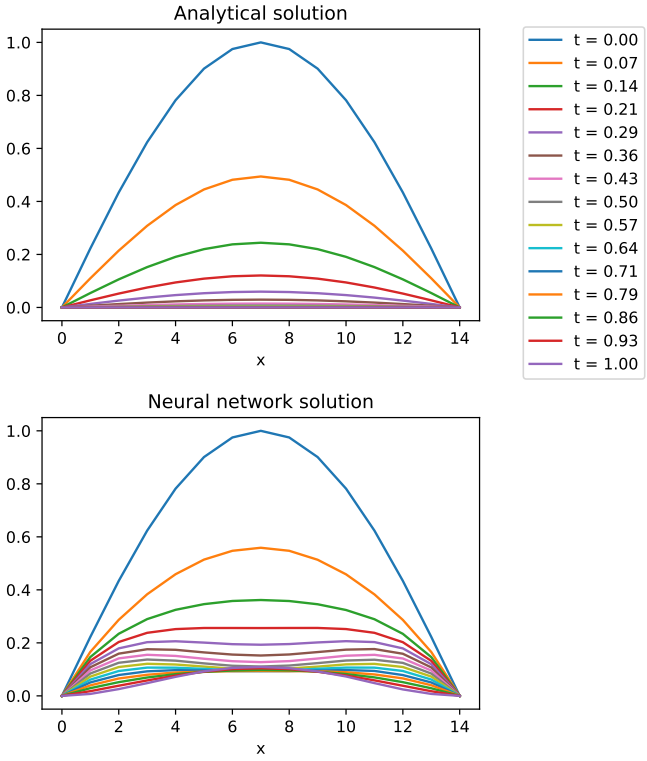

FIG. 2.— Analytical and neural net solutions to the heat flow equation

The maximum output error, comparing to the analytical solution (2), is 0.16 after 50000 epochs. The network struggles particularly for t close to 1, and this is the case for all choices of hyperparameters that have been tried. The algorithm stagnates at around an error of 0.16 or higher for all the tested hyperparameters. The Forward Euler solution has a max absolute error of 0.027, which is significantly better. Because of the large amount of possible hyperparameters, it is hard to conclude whether the fault lies with the method itself or that better results could be obtained with other hyperparameters.

3.2. *Eigenvalue problem*

The network manages to produce correct eigenvalues for all the randomly generated symmetric matrices that were tested. Generally the network converges to the correct solution within a 10^{-4} error margin within the three hundred first epochs.

4. CONCLUSION

The neural network manages to produce a decent approximation to the differential equation, but for these choices of hyperparameters is still outperformed by other methods. As mentioned, the large amount of possibilities for hyperparameter tuning leads me to believe that a network could be able to solve the problem, but more work has to be done to find the right parameters. The network struggles to predict solutions that are close to constant in x, i.e. for t-values closer to one. Freezing the some layers after a period of training and continuing to iterate for higher t-values could be an idea to improve these solutions. The network is easily able to produce correct eigenvalues for 6×6 matrices, but seems to be a quite heavy handed way of doing a quite simple operation that is easily managed for instance by direct gradient descent on the Rayleigh quotient. However, given the flexibility of neural nets in solving a different range of problems, there is a lot of promise in developing techniques that solve more complex eigenvalue problems and differential equations. 'Out of the box' methods that are flexible in their applications are in high demand, and neural nets is good starting point for developing such methods.

5. REFERENCES

Hein, Kristine Baluka (2018), *Data Analysis and Machine Learning: Using Neural networks to solve ODEs and PDEs*