Project 3 on Machine Learning: Solving partial differential equations with neural networks

Bjørn Grønntun

December 15, 2019

Abstract

This is rather abstract.

1 Heat equation - physical interpretation and analytical solution

If we have a rod of length $L = 1$ with no lateral temperature loss, the temperature in the interior points of the rod can be modelled by a function $u(x, t)$, where $x \in (0, 1)$ is the distance from left point of the rod and t is time. It is well established that the interior temperature will conform to the heat equation

$$u_t = \alpha^2 u_{xx} \tag{1}$$

For simplicity, we will let $\alpha = 1$. As boundary condition, we will impose that the endpoints of the rod be kept at temperature 0. Initially, the temperature profile is given by $u(x, 0) = \sin(\pi x)$. Thus, the intitial-boundary value problem we are going to study is as follows:

$$u_t = u_{xx} \tag{2}$$

$$u(0, t) = u(1, t) = 0 \tag{3}$$

$$u(x, 0) = \sin(\pi x) \tag{4}$$

The analytical solution to this problem can be found by the classical technique of separation by variables. Even if this derivation is given in every text on PDEs, we now proceed to give a quick sketch of this method. To start with, we look for solutions of the form $u(x, t) = X(x)T(t)$. This substitution transforms the PDE to $X(x)T'(t) = X''(x)T(t)$, which means that

$$\frac{T'(t)}{T(t)} = \frac{X''(x)}{X(x)} \tag{5}$$

As the left side of this equation depends solely on t and the right side depends solely on x, the two sides must be equal to a common constant. Furthermore, this constant must be negative. (Explanation). This leads to the two ODEs

$$T'(t) + \lambda^2 T(t) = 0 \tag{6}$$

and

$$X''(x) + \lambda^2 X(x) = 0 \tag{7}$$

These equations are very easy to solve. (6) has the general solution

$$T(t) = Ce^{-\lambda^2 t} \tag{8}$$

whereas (7) has the general solution

$$X(x) = D\sin(\lambda x) + E\cos(\lambda x) \tag{9}$$

Here, C, D and E are arbitrary constants. Thus the PDE is satisfied by all functions of the form

$$u(x,t) = e^{-\lambda^2 t}[A\sin(\lambda x) + B\cos(\lambda x)] \tag{10}$$

where A and B are arbitrary constants.

Now, we want to bring in the boundary conditions, the first of which is $u(0,t) = 0$. Putting $x = 0$ in (10), we obtain $Be^{-\lambda^2 t} = 0$, so B must be equal to 0. The second boundary condition, $u(1,t)$, now yields

$$Ae^{-\lambda^2 t}\sin(\lambda) = 0 \tag{11}$$

As can be seen, we must either have $A = 0$, leading to the trivial solution $u \equiv 0$, or $\sin(\lambda) = 0$. Thus, λ has to be an integer multiple of π. This leads to an entire family of solutions satisfying both the PDE and the boundary conditions. This family is given by

$$u_n(x,t) = A_n e^{-(n\pi)^2 t}\sin(n\pi x) \tag{12}$$

where $n \in \mathbb{Z}^+$.

It is easily verified that any linear combination of solutions satisfying the PDE and the boundary conditions will again be a solution. This leads us to consider solutions of the form

$$u(x,t) = \sum_{n=1}^{\infty} A_n e^{-(n\pi)^2 t}\sin(n\pi x) \tag{13}$$

In order that the solution satisfy the initial condition $u(x,0) = \sin(\pi x)$, we put

$$u(x,0) = \sin(\pi x) \tag{14}$$

that is

$$\sum_{n=1}^{\infty} A_n sin(n\pi x) = \sin(\pi x) \tag{15}$$

In order to satisfy this equation, we simply let $A_1 = 1$ and $A_n = 0$ for $n \neq 1$. Now we finally have reached our analytical solution:

$$u(x,t) = e^{-\pi^2 t} \sin(\pi x) \tag{16}$$

It is easily verified that this solution satisfy the PDE, the boundary conditions and the initial condition. It can be shown that the solution is unique.

2 Numerical methods - overview

We seek a bivariate function, defined on $[0,1] \times \mathbb{R}^+$. Of course, in a numerical setting we have to constrain the function domain somewhat: We choose to construct a grid of points on $[0,1] \times [0,1]$. If there are $n+1$ grid lines parallell to the t axis and $m+1$ grid lines parallell to the x axis, the distance between the grid lines when we move in the x direction is $h = 1/n$ and the distance between the grid lines when we move in the t direction is $k = 1/m$. The values for x and t will thus be given by

$$x_j = jh \tag{17}$$

where $j = 0, 1, ..., n$
and

$$t_i = ik \tag{18}$$

where $i = 0, 1, ..., m$.

3 Finite difference scheme - theory

In order to construct a finite-difference scheme to solve the heat equation, we use truncated forms of the Taylor series for u to obtain the following approximations:

$$u_t = \frac{1}{k}[u(x, t+k) - u(x,k)] \tag{19}$$

and

$$u_{xx} = \frac{1}{h^2}[u(x+h,t) - 2u(x,t) + u(x-h,t)] \tag{20}$$

By using the notation $u_{i,j}$ for $u(x_j, t_i)$, we can write these equations as

$$u_t = \frac{1}{k}[u_{i+1,j} - u_{i,j}] \tag{21}$$

and

$$u_{xx} = \frac{1}{h^2}[u_{i,j+1} - 2u_{i,j} + u_{i,j-1}] \tag{22}$$

The form of the heat equation we are studying demands that these be equal to each other. Solving for $u_{i+1,j}$, we obtain

$$u_{i+1,j} = u_{i,j} + \frac{k}{h^2}[u_{i,j+1} - 2u_{i,j} + u_{i,j-1}] \tag{23}$$

We now have a solution at each time step in terms of the solution at the previous time step. This gives us the following algorithm:

4 Finite difference scheme - implementation and results

The above algorithm is implemented in `src/difference.py`. The testing is in `notebooks/solving-pde.ipynb`. In order to meet the stability criterion for the forward method, we begin by constructing a 11×201 grid, corresponding to step sizes of $\Delta x = 0.1$, $\Delta t = 0.005$. This just meets the stability criterion, since

$$\frac{\Delta t}{\Delta x^2} = \frac{0.005}{0.01} = 0.5 \tag{24}$$

In order to obtain even better approximations, we also construct a 101×20001 grid, corresponding to step sizes of $\Delta x = 0.01$, $\Delta t = 0.00005$. Again, this just meets the stability criterion, since

$$\frac{\Delta t}{\Delta x^2} = \frac{0.00005}{0.0001} = 0.5 \tag{25}$$

In order to evaluate how well our numerical solutions approach the analytical solution, we look at the maximal value of the absolute differences between the two. We find that when using the 11×201 grid, we obtain a maximal absolute difference of 0.006, whereas when using the 101×20001 grid, the maximal absolute difference is reduced to $6.05 \cdot 10^{-5}$.

5 Neural networks

We now turn to a completely different method for obtaining numerical solutions of differential equations. A neural network should in principle be able to approximate the values of a function on a given input grid. We just have to make sure that the solutions proposed by our method actually fulfill the initial conditions and boundary conditions for our problem. How this is done is somewhat problem dependent, but it is usually possible to formulate a trial solution of the type

$$g_{\text{trial}}(\mathbf{x}, t) = h_1(\mathbf{x}, t) + h_2(\mathbf{x}, t, N(\mathbf{x}, t, P)) \tag{26}$$

Here, h_1 depends on the inputs $\mathbf{x}, t$ only, whereas h_2 is allowed to depend on the outputs of the outputs of the neural network. The state of the neural network is given by P, which represents its weights and biases. The role of h_1 and h_2 is to make sure any solution respect the initial and boundary conditions. In our case, we choose

$\mathrm{g}_{\mathrm{trial}}(x,t) = (1-t)\sin(\pi x) + x(1-x)tN(x,t,P)$

It is easily verified that $g_{\mathrm{trial}}(x,0) = \sin(\pi x)$ for any value of x, meaning that any solution will conform to our initial condition. Moreover, we see that $g_{\mathrm{trial}}(0,t) = g_{\mathrm{trial}}(1,t) = 0$ for any value of t, which means that our solutions always respect our boundary conditions.

We train our neural network by evaluating how well our trial solutions solve the PDE. As we want our solutions to solve

$$u_t = u_{xx} \tag{27}$$

that is

$$u_t - u_{xx} = 0 \tag{28}$$

we use as our cost function $C = \mathrm{MSE}((g_{\mathrm{trial}})_t - (g_{\mathrm{trial}})_{xx}, 0)$ and use this to train our network, thus obtaining gradually better approximations to the true solution.

Our Tensorflow implementation of this method is tested in `notebooks/solving-pde.ipynb`.

6 Comparison of methods

7 Eigenvalue problems

Given a matrix $A \in \mathbb{R}^{n\times n}$, an *eigenpair* $(\lambda, x), x \neq 0$ is a solution of the equation

$$Ax = \lambda x \tag{29}$$

Here, the scalar λ called an *eigenvalue* and $x \in \mathbb{R}^n$ is a non-zero vector. Here, we only consider real symmetric matrices. It is easily proved that any eigenvalue of a real symmetric matrix is a real number. Finding eigenvectors and eigenvalues of such matrices is an important problem in engineering, and several numerical methods have been proposed. We here propose a simple method employing neural networks. Given any vector $x \in \mathbb{R}^n$, the corresponding Rayleigh quotient for a real symmetric matrix A is defined as

$$R_A(x) = \frac{x^T A x}{x^T x} \tag{30}$$

A trivial fact is that if x is an eigenvector, its Rayleigh quotient will be equal to the corresponding eigenvalue. Moreover, it can be shown that the Rayleigh quotient is maximized and minimized by two of matrix' eigenvectors. By maximizing or minimizing the Rayleigh quotient, we should therefore be able to

find the eigenvectors corresponding to the largest and smallest eigenvalue. It is then a simple matter to find the eigenvalues using the Rayleigh quotient. The function `eigen` in `src/eigenvectors.py` works in the following way: A random vector is inputed into a dense neural network, yielding a proposed eigenvector as output. We compute the corresponding Rayleigh quotient and use gradient descent to update the weights of the network, thus reducing the Rayleigh quotient. This process is iterated until the reduction of the Rayleigh quotient in subsequent iterations gets below a certain tolerance, at which point we stop the process.

In the notebook `notebooks/finding-eigenvalues.ipynb` we use our implementation to find unit eigenvectors of a 6×6 real symmetric matrix. We find that a neural net with 5 input nodes and two hidden layers, each with 5 nodes and tanh activation function, perfoms quite well. Comparison with the eigenvectors found by `numpy.linalg.eig` yields a mean squared deviation of $1.057 \cdot 10^{-13}$ for one eigenvector and $1.33 \cdot 10^{-11}$ for the other.

8 Wrapping up