Solving ordinary differential with Using a hybrid neural network

Abstract

Along this project, feed forward artificial neural network algorithm is been applied for estimating differential equations. While robust training and backpropagation development is not implemented, by using Newton Raphson approximation for backpropagation development the algorithm provided highly accurate and stable method for solving ordinary differential equation. Step wisely talk, hyperbolic secant is defined as activation function of modified feed forward neural network and build an algorithm for approximation of differential equation answers. Some conditions like initial values and boundary conditions are required for training our model inside given interval then using model for providing algorithm that can return almost same response of analytical answer of the ordinary differential equation.

Definitions and theorems

Activation function (transformer): A function used to transform the activation level of a unit (neuron) into an output signal. Typically, activation functions have a "squashing" effect. There are variety of choices of activation functions, the famous example is sigmoid function with range of (0, 1) and order of continuity (smoothness) of C^{∞} (Infinity). In this project, the Activation function is hyperbolic secant, which hold the same characteristics.

$$S(n) = \frac{2}{e^{-n} + e^{n}}$$

Learning process used in this feed forward neural networks follows law of error propagation (BP). This law is about error correction learning rule, therefore, to calculate sensitivities for different layers of neurons, numerous layers of derivatives of conversation neurons functions is required. So, having the function like hyperbolic secant as one of the sigmoid family of functions with capability of infinitely many derivatives in complex space make more sense. The relation between middle layer (hidden layer) and output layer is linear which make us to have only one hidden layer for our purpose, by these considerations, we are having one input layer and one middle layer (hidden layer) and one output layer. From Kolmogorov existence theorem $\{(v_{t_1}, \ldots, v_{t_k}\ (F_1 \times \ldots \times F_k) = P(X_{t_1} \in F_1, \ldots, X_{t_k} \in F_k)\}$, we know that for these three layer perceptron with n(2n+1) nodes can compute any continuous function of n variables. Accuracy of the approximation depends on the number of layers of hidden layers. The multi-layered feed forward network is training problem, in some cases this network can be ill-conditioned equation matrices, but in this case, it will not create ill-condition metrics. Here we use forward neural network to form a model (function of many variables) from their training data (input data). We let the network learn from the theory of differential equations, in order to produce a function consisting of adjustable parameters originally came from neural networks. By minimizing this function with many variables, we gain the optimum adjustable parameters for the corresponding mutli-layered perceptron. This analytic closed form function is particularly useful for computing the solution of differential equation. Simply we can say that this is an approximation function to the corresponding differential equation solutions. The hybrid technique adapted here for computing the solution of differential equation is different with neural network, but nicely uses neural network to produce a nonlinear function, namely energy function, by minimizing of this function which is guaranteed to converge. thanks to the optimum structure of the non-linear function.

Problem Formulation:

For our concern we can have this general initial / boundary value problems in form below: Where D is the differential operator of degree n and C is an initial / boundary operator. Here x is independent variable and y is unknown dependent variable to be calculated.

$$\left\{\begin{aligned} D\left[x, y(x), \frac{dy}{dx}, \frac{d^2y}{dx^2}, \dots, \frac{d^ny}{dx^n}\right] = 0 \\ C\left[x, y(x), \frac{dy}{dx}, \frac{d^2y}{dx^2}, \dots, \frac{d^ny}{dx^n}\right] = 0 \end{aligned}\right\}, \quad x \in [a, b]\; x = a\,, and/\,or\; b,$$

Let assume that a general approximation solution to (1) is in the form $y_t(x, P)$ for y_t as a dependent variable on x and P, where P is an adjustable parameter involving weight and biases in the structure of the three-layer feed forward neutral network as mentioned previously. The trial solution y_t is an approximation solution to y (with respect to some norm) for the optimized values of unknown points in [a, b] is equivalent to calculate the functional $y_t(x, P)$ that satisfies the following constrained optimization problem (Kincaid 2002):

$$(1)\quad \left\{\begin{array}{c} Min \int_a^b \left\| D\left[x, y_t(x, P), \frac{dy_t(x, P)}{dx}, \frac{d^2y_t(x, P)}{dx^2}, \dots, \frac{d^ny_t(x, P)}{dx^n}\right]\right\| dx \\ C\left[x, y_t(x, P), \frac{dy_t(x, P)}{dx}, \frac{d^2y_t(x, P)}{dx^2}, \dots, \frac{d^ny_t(x, P)}{dx^n}\right] = 0 \;\; x \in [a, b]\; x = a\,, and/\,or\; b, \end{array}\right.$$

Where we use ||. || for Euclidian norm. Because of integrals' close association with the sigma, and because we are seeking for finding a function $y_t(x, P)$ amoung all those function belong to continuously differentiable function on [a, b] that vanishes x= a and x= b, the problem summarized in

$$(2)\begin{cases} Min\sum_{1}^{m}\left\{D\left[x, y_t(x,P), \frac{dy_t(x,P)}{dx}, \frac{d^2y_t(x,P)}{dx^2}, \dots, \frac{d^ny_t(x,P)}{dx^n}\right]\right\}^2 \\ C\left[x, y_t(x,P), \frac{dy_t(x,P)}{dx}, \frac{d^2y_t(x,P)}{dx^2}, \dots, \frac{d^ny_t(x,P)}{dx^n}\right] = 0 \;\; x \in [a,b] \; x = a\,, and/\,or\; b, \end{cases}$$

Where $\{x_i\}_{i=1}^{m}$ are some arbitrary collocation points for the interval [a, b] for a discrete least square norm. This is mesh free discrete constrained minimization problem for the (1) .In Order to transfer problem (2) into an unconstrained optimization problem that is simpler to deal with, we define the trial solution to be in the following form:

$$y_t(x,P) = \alpha(x) + \beta[x, N(x,P)]$$

Where the first term in the right-hand side does not involve with adjustable parameters and satisfies the initial values or boundary, in case of boundary value problem. The second term in the right-hand side is a feed forward three-layer perceptron consisting of an input x and the output $N(x,P)$. In (2) minimization can considered as the training process for the proposed neural network. The corresponding error with every entry x is E(x) that has to be minimized. This residual is the consequence of direct substitution of trial function $y_t(x,P)$ instead of Y(x) into the problem (1). E(x) clearly measure the amount by which the exact solution y(x) do not satisfy the differential equation. In order to compute the residual, one may need to calculate the output $N(x,P)$ together with its derivatives with respect to x as an entry variable.

Let consider a three-layered perceptron ANN (artificial neural network) with one unit entry x, one hidden layer consisting H hyperbolic secant function (sigmoid family) and one unit output N(x, p) . For every entry x the perceptron's output is $\sum_{i=1}^{H} v_i s(z_i)$ in which $z_i = w_i Q(x) + b_i$ and $w_i$is a weight parameter from input layer to the i th hidden layer v_i is an i th weight parameter from hidden layer to the output of the i th hidden unit and $s(z_i)$ is an arbitrary activation function (sigmoid in most cases).We define the $Q(x)$ which is as below.

$Q(x) = (x+1)\varepsilon \qquad \varepsilon = 0.4$

By mentioned considerations we need to take a look at the derivatives of N(x, P);

$$\frac{d^kN}{dx^k} = \sum_{j=1}^{H} v_j w^k s'(z_j)$$

$$\frac{d^ny_t(x,P)}{dx^n} = \sum_{j=1}^{H} v_j w^n s^{(n)}(z_j)$$

$$\sum_{j=1}^{H} v_j w^k s'(z_j) = \sum_{j=1}^{H} v_j w^k \frac{-2(e^{w_jQ(x)+b_j} + e^{-w_jQ(x)+b_j})}{(e^{w_jQ(x)+b_j} + e^{-w_jQ(x)+b_j})^2}$$

By putting all together for initial boundary problem:

$$\begin{cases} \dfrac{dy}{dx} = f(x,y), \quad x \in [a,b] \\ \qquad\quad y(a) = A \end{cases}$$

$$(4) \quad y_T(x,P) = A + (x-a)N(x,P)$$

$$E(p) = \sum_{i=1}^{M} \left\{ \frac{dy_t(x,P)}{dx} - f[x_i, y_t(x,P)] \right\}^2$$

Where $\{x_i\}_{i=1}^{m}$ are discrete points belongs to the interval [a, b]. Then differentiating from trial function $y_t(x,P)$ in (4) we obtain:

$$\frac{dy_T(x,P)}{dx} = N(x,P) + (x-a)\frac{dN(x,P)}{dx}$$

Now we can construct our energy function for first order differential equation:

$$E(p) = -\sum_{i=1}^{M} \left\{ (x_i - a) \sum_{1}^{H} v_j w_j \frac{-2\left(e^{w_j Q(x)+b_j} + e^{-w_j Q(x)+b_j}\right)}{\left(e^{w_j Q(x)+b_j} + e^{-w_j Q(x)+b_j}\right)^2} - \sum_{1}^{H} v_i w_i \frac{-2}{\left(e^{w_j Q(x)+b_j} + e^{-w_j Q(x)+b_j}\right)} + f[x_i, y_t(x,P)] \right\}^2$$

Minimization method will be BFGS quasi-Newton with fminunc minimizer in MatLab

Numerical examples:

$$\begin{cases} \dfrac{dy}{dx} = 4x^3 - 3x^2 + 2, \quad x \in [0,1] \\ \qquad\quad y(0) = 0 \end{cases}$$

The exact solution is: $y(x) = x^4 - x^3 + 2x$

$$y_t(x,P) = 0 + (x-0) \sum_{j=1}^{H=5} v_i w_i \frac{-2}{\left(e^{w_j Q(x)+b_j} + e^{-w_j Q(x)+b_j}\right)}$$

$$y_t(x,P) = x\sum_{j=1}^{H=5} v_j w_j \frac{-2}{\left(e^{w_jQ(x)+b_j} + e^{-w_jQ(x)+b_j}\right)}$$

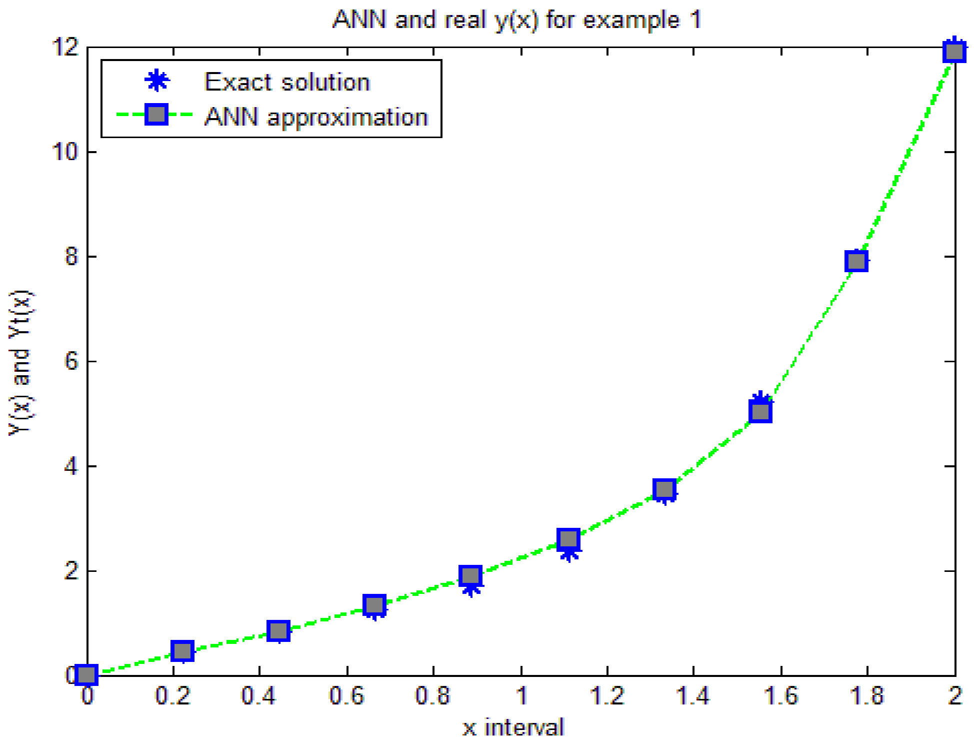
$$E(p) = \sum_{i=1}^{M}\left\{(x_i)\sum_{j=1}^{H} v_j w_j \frac{-2\left(e^{w_jQ(x_i)+b_j} + e^{-w_jQ(x_i)+b_j}\right)}{\left(e^{w_jQ(x_i)+b_j} + e^{-w_jQ(x_i)+b_j}\right)^2} - \sum_{j=1}^{H} v_j w_j \frac{-2}{\left(e^{w_jQ(x_i)+b_j} + e^{-w_jQ(x_i)+b_j}\right)} + 4{x_i}^3 - 3{x_i}^2 + 2\right\}^2$$

Answers for three unknowns for network v_i , w_i , b_i $i = 1,2,3,4,5$

i	1	2	3	4	5
v_i	-2.3788	-1.4642	3.0694	1.3628	3.1620
w_i	2.6813	-3.3218	-0.7165	-2.4613	-12.1151
b_i	7.7039	1.8148	1.3142	0.9137	14.6638

For purpose of Analysis of Method, I chose 10 points between [0, 2] rather than [0,1] therefor points are equivalent in distance but they follow different mesh size and interval so there is no overfitting or instability

i	1	2	3	4	5	6	7	8	9	10
x_i	o	0.2222	0.4444	0.6667	0.8889	1.1111	1.3333	1.5556	1.7778	2.000
y_i	0	0.4359	0.8401	1.2346	1.6997	2.3746	3.4568	5.2023	7.9256	12.00
$y_{t\ i}$	0	0.4253	0.8407	1.3072	1.8766	2.5842	3.5142	4.9842	7.8810	11.8674



Another Example:

$$\begin{cases} \dfrac{dy}{dx} = e^{x} & x \in [0,1] \\ y(0) = 1 \end{cases}$$

Exact solution: $y(x) = e^{x}$

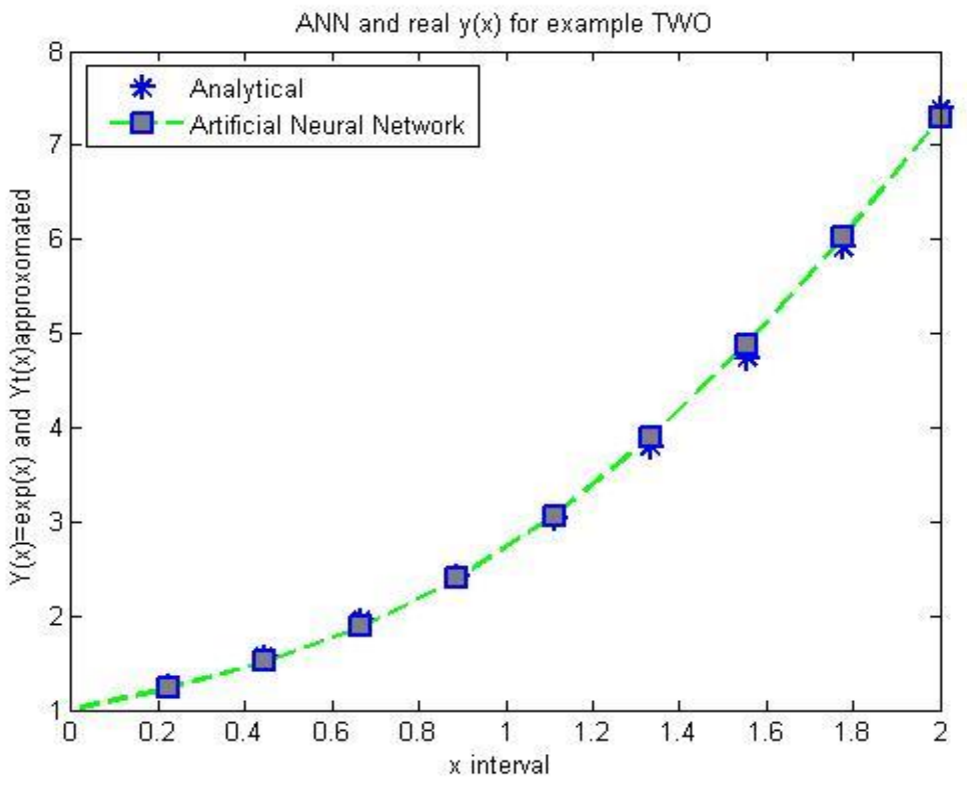
$$y_t(x,P) = 1 + x \sum_{j=1}^{H=5} v_j w_j \frac{-2}{\left(e^{w_j Q(x)+b_j} + e^{-w_j Q(x)+b_j}\right)}$$

$$E(p) = \sum_{i=1}^{M=6} \left\{ (x_i) \sum_{j=1}^{H} v_j w_j \frac{-2\left(e^{w_j Q(x_i)+b_j} + e^{-w_j Q(x_i)+b_j}\right)}{\left(e^{w_j Q(x_i)+b_j} + e^{-w_j Q(x_i)+b_j}\right)^2} + \sum_{j=1}^{H} v_j w_j \frac{-2}{\left(e^{w_j Q(x_i)+b_j} + e^{-w_j Q(x_i)+b_j}\right)} + e^{x_i} \right\}^2$$

Answers for example two:

i	1	2	3	4	5
v_i	-0.5307	5.9447	5.5733	3.1224	-5.2659
w_i	-0.1169	-1.2770	1.4320	3.1183	-0.8362
b_i	6.2674	1.9431	0.6332	2.0201	-0.0003

i	1	2	3	4	5	6	7	8	9	10
x_i	o	0.2222	0.4444	0.6667	0.8889	1.1111	1.3333	1.5556	1.7778	2.000
y_i	1.0	1.2488	1.5596	1.9477	2.4324	3.0377	3.7937	4.7377	5.9167	7.3891
$y_{t\ i}$	1.0	1.2309	1.5124	1.8889	2.3964	3.0599	3.8914	4.8881	6.0310	7.2351



Conclusion:

For example, Energy function E(P), 3.6389e-06 is achieved, because only 6 points is used for training and then plot for 10 point from training as well for extra same period of [1,2]. By considering different mesh size and twice as training boundary, to have out of sample performance.

For the 2nd example same 6 points of training in interval [1,0] is used, afterward extrapolated on [0,2] for 10 points, again different mesh size and earn a good energy function size of E(P) = 2.6061e-07.

Codes

Example 1:

Energy Function:

```
function E = adjfcn(X)
v=X(:,1);
w=X(:,2);
b=X(:,3);
M=6;
x=linspace(0,1,M);
E=0;
for i=1:M
H=5;
y=0;S=0;
for j=1:H
    q(i)=(x(i)+1)*(0.4);
    n=w(j)*q(i)+b(j);
    S =exp(n)+exp(-n);
   y= y+(v(j)*(2/S) +x(i)*v(j)*w(j)*(-2*S/(S)^2));
end

E=E+(y-4*x(i)^3+3*x(i)^2-2)^2;
End
```

Network Function

```
function [Y,Yt] = ExactSolution(V)

v=V(:,1); w=V(:,2); b=V(:,3);M=10;
x=linspace(0,2,M);
Yt=0;
for i=1:M
H=5;
;y=0;Se=0;
for j=1:H
    q(i)=(x(i)+1)*(0.4);
    n=w(j)*q(i)+b(j);
    Se =Se+(v(j)*(2/(exp(n)+exp(-n))));
end
Yt(i)=Se*x(i);
Y(i)= x(i)^4-x(i)^3+2*x(i);
end
```

```
Error=max(abs(Yt-Y));
plot (x,Y,'r*',x,Yt,'--gs',...
    'LineWidth',2,...
    'MarkerSize',10,...
    'MarkerEdgeColor','b',...
    'MarkerFaceColor',[0.5,0.5,0.5]);
legend('Exact solution','ANN approximation','location','NorthWest')
title(' ANN and real y(x) for example 1 ')
xlabel('x interval ')
ylabel('Y(x) and Yt(x)')
```

Optimization

```
opt = optimoptions(@fminunc,'Algorithm','quasi-newton');
X1=rand(5,3);
[V,fval,exitflag,output]=fminunc(@adjfcn,X1,opt);
ExactSolution(V);
```

Example 2:

Energy Function

```
function E = AFN(X)
v=X(:,1);
w=X(:,2);
b=X(:,3);
x=[0.0,0.2,0.4,0.6,0.8,1];
E=0;M=6;
for i=1:6
H=5;
;y=0;S=0;
for j=1:H
    q(i)=(x(i)+1)*.4;
    n=w(j)*q(i)+b(j);
    S =(2/(exp(n)+exp(-n)));
   y= y+v(j)*S +x(i)*v(j)*w(j)*(S-S^2);
end
Y=exp(x(i));
E=E+(y-Y)^2;
End
```

Network Function:

```
function [Y,Yt] =  EXPReal(V)
v=V(:,1); w=V(:,2); b=V(:,3);
Yt=0;M=10;
x=linspace(0,2,M);
for i=1:M
H=5;
;y=0;S=0;
for j=1:H
    q(i)=(x(i)+1)*(0.4);
    n=w(j)*q(i)+b(j);
    S =S+v(j)*(2/(exp(n)+exp(-n)));
end
```

```
Yt(i)=1+S*x(i);
Y(i)= exp(x(i));
end
plot (x,Y,'r*',x,Yt,'--gs',...
    'LineWidth',2,...
    'MarkerSize',10,...
    'MarkerEdgeColor','b',...
    'MarkerFaceColor',[0.5,0.5,0.5]);
legend('Analytical','Artificial Neural Network','location','NorthWest')
title(' ANN and real y(x) for example TWO ')
xlabel('x interval ')
ylabel('Y(x)=exp(x) and Yt(x)approxomated ')
end
```

Optimization

```
opt = optimoptions(@fminunc,'Algorithm','quasi-newton');
X1=rand(5,3);
[V,fval,exitflag,output]=fminunc(@AFN,X1,opt);
```

References

[1] Dissanayake M. W. M. G., Phan-Thien N., Neural-network-based approximations for solving partial differential equations, Communications in Numerical Methods in Engineering, 10 (1994) 195-201.

[2] Hornick K., Stinchcombe M., White Multilayer feedforward networks are universal approximators, Neural Networks, 2 (1989) 359-366.

[3] Lee H., Kang I.S., Neural algorithms for solving differential equations, journal of computational physics, 91(1990) 110-131.

[4] Liu B., Jammes B., Solving ordinary differential equations by neural networks, in: Proceeding of 13th European Simulation Multi-Conference Modelling and Simulation: A Tool for the Next Millennium, Warsaw, Poland, June 14, (1999).

[5] Lagaris I. E., Likas A., Fotiadis D. I., Artificial neural networks for solving ordinary and partial differential equations, IEEE Transactions on Neural Networks, 9 (5) (1998) 987-1000.

[6] Liu C., Nocedal J., On the limited memory BFGS method for large scale optimization. Mathematical Programming, 45(3) (1989) 503-528.

[7] Meade Jr A.J., Fernandez A.A., The numerical solution of linear ordinary differential equations by feed forward neural networks, Mathematical and Computer Modelling, 19 (12) (1994) 1-25.

[8] Meade Jr A.J., Fernandez A.A., Solution of nonlinear ordinary differential equations by feedforward neural networks, Mathematical and Computer Modelling, 20 (9) (1994) 19-44.

[9] Malek A., Shekari R., Numerical solution for high order differential equations, using a hybrid neural networkOptimization method, Applied Mathematics and Computation, 183 (2006) 260-271.

[10]. D.R. Kincaid, E.W. Cheney, Numerical Analysis: Mathematics of Scientific Computing, third ed.,Brook/Cole, Pasific, CA 2002

[11] Ezadi S., Parandin N., An Application of Neural Networks to Solve Ordinary Differential Equations, International Journal of Mathematical Modelling & Computations Vol. 03, No. 03, 2013, 245- 252

[12] https://en.wikipedia.org/wiki/Hopfield_network

[13] https://en.wikipedia.org/wiki/Activation_function.