Neural Modeling and Differential Equations

I. E. Lagaris

CSE, University of Ioannina

Abstract

The universal approximation capability of neural networks is exploited to recover solutions of DEs.
The process of solving a DE is reduced to that of training a neural form.
Boundary conditions may be satisfied either by proper construction of the neural form, or as constraints in an optimization setting.

Talk Structure

Important related results

Kurt Hornik, Maxwell B. Stinchcombe, Halbert White, 1989

Multilayer feedforward networks are universal approximators.

George Cybenko, 1989

Approximation by Superpositions of a Sigmoidal Function.

In these articles it is proved that single hidden layer networks, with sigmoid activation functions, can approximate to any desired degree of accuracy any function, provided that sufficient number of neurons are available. This result was later extended to Gaussian RBF networks as well.

The above results are based on the work of
Andrey N. Kolmogorov, 1957

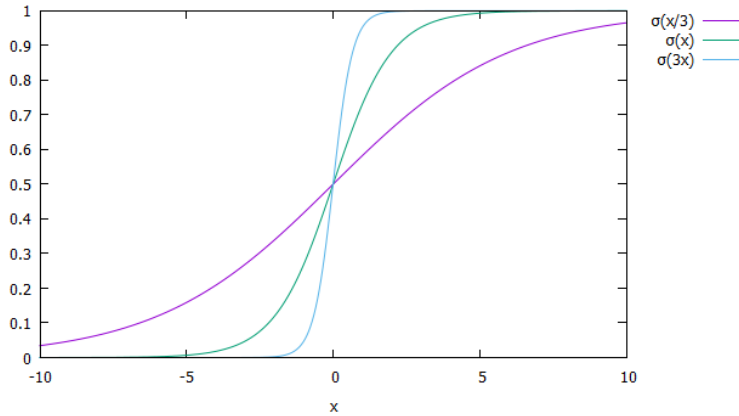
On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition.

Common Expressions of ANNs

Sigmoidal Percepton:

$$N(x,w) = \sum_{k=1}^{Nodes} a_k \sigma(p_k^T x + b_k)$$

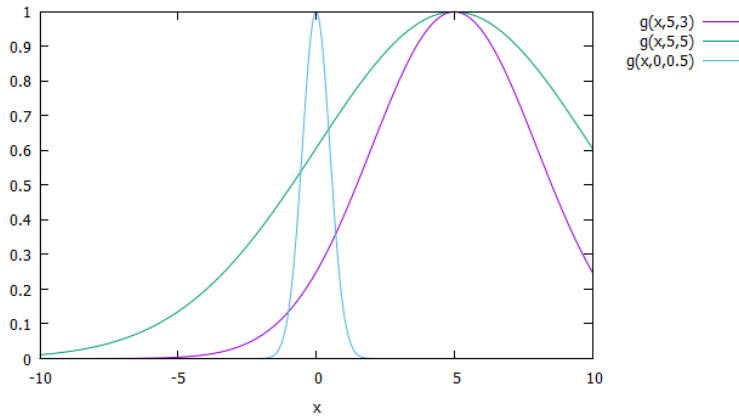
$$w = \{a, p, b\}, \text{ and } \sigma(z) = \frac{1}{1+\exp(-z)}$$



Gaussian RBF:

$$N(x,w) = \sum_{k=1}^{Nodes} a_k g(x, \mu_k, s_k)$$

$$w = \{a, \mu, s\}, \text{ and } g(x,\mu,s) = e^{-\frac{1}{2}\left(\frac{||x-\mu||}{s}\right)^2}$$



Introduction to training

Given a set of data:

$$S = \{x_i, y_i\},\ x_i \in R^d,\ y_i \in R$$

where y_i, is generated by a function $y(x)$ as: $\boldsymbol{y_i = y(x_i)}$,
Neural Networks (NN) are used to approximate (or "learn")
the underlying function $y(x)$, by considering only the information
contained in the given dataset S.
A parametric model is constructed:

$$Y_M(x, w) = N(x, w),\quad N(x, w) \text{ being a NN}$$

and then the Mean Squared Error (MSE):

$$E(w) = \frac{1}{\#S} \sum_{x_i \in S} \left[Y_M(x_i, w) - y_i \right]^2$$

is minimized wrt the parameters w, the NN "weights".

Is it any different from Data-Fitting ?

The technique is identical, i.e. minimization of the MSE.
The important modeling philosophy is radically different.

In data-fitting the model used is based on theoretical considerations, phenomenology, or even on intuition. For example, when fitting the nucleon-nucleon phase shifts to construct a two-body nuclear potential, the long-range tensor force is attributed to the one-pion exchange process, imposing a Yukawa term, that is proportional to:

$$\left(1+\frac{3}{\mu r}+\frac{3}{(\mu r)^2}\right)\frac{e^{-\mu r}}{\mu r}, \quad \mu \text{ being the mass of pion}$$

This being so specific, is biased and very restrictive.

Neural Networks **do not Impose**; **Discover**

Neural Networks are:

- "**Flexible**" enough, that actually do not impose any specific form. (Essentially model independent).
- **Unbiased** estimators of any function.
- Exclusively **data-driven**.

NNs Discover the underlying data-generating "law",
by **"learning"** from the data.

Neural Forms may Impose properties

Definition:

A **Neural Form** (NF), is an expression that involves a Neural Network.

Examples: $A(x) + Z(x)N(x,w)$, $B(x) + C(x)\dfrac{dN(x,w)}{dx}$, $\cdots$

- If the NF: $Y_M(x,w) = a + xN(x,w)$, is used to learn the data in S, then the property: $Y_M(0,w) = a$, **is exactly satisfied.**

- Similarly, the NF: $Y_M(x,w) = a + (b-a)x + x(x-1)N(x,w)$, **satisfies exactly:** $Y_M(0,w) = a$, **and** $Y_M(1,w) = b$.

Neural Forms play an **important** role in the **model construction** process !!!

Differential Equations

The solution of a DE can be represented by a Neural Form.
Consider the simple ODE:

$$\frac{d}{dx}\Psi(x) = f(x), \quad x \in [0,1]$$

with the Boundary Condition: $\Psi(0) = y_0$

Use for the trial solution a Neural Form, that satisfies the BC:

$$\Psi(x) \approx \Psi_t(x,w) = y_0 + xN(x,w)$$

Substituting the trial solution in the DE yields:

$$x\frac{d}{dx}N(x,w) + N(x,w) = f(x), \quad \forall\, x \in [0,1]$$

The Solution Procedure

Let $x_i, \forall\, i = 1, 2, \cdots, N$, be a mesh in $[0, 1]$.

Hence, $\forall\, i = 1, 2, \cdots, N$, we require:

$$x_i \frac{d}{dx} N(x_i, w) + N(x_i, w) = f(x_i)$$

To satisfy this requirement, the procedure followed is to minimize:

$$E(w) = \frac{1}{N} \sum_{i=1}^{N} \left[x_i \frac{d}{dx} N(x_i, w) + N(x_i, w) - f(x_i) \right]^2$$

Equivalent to learning the set $\{x_i, f(x_i)\}$, using the Neural Form model :

$$Y_M(x, w) = x \frac{dN}{dx}(x, w) + N(x, w)$$

Boundary Conditions I

For 2^{nd} order ODEs, two conditions are required.
Consider the two-point BCs, $\boldsymbol{x} \in [\boldsymbol{a}, \boldsymbol{b}]$.

Examples: $\boldsymbol{\Psi(a) = y_1}$ **and** $\boldsymbol{\Psi(b) = y_2}$, or: $\boldsymbol{\Psi(a) = y_1}$ **and** $\boldsymbol{\Psi'(b) = y'_2}$

The trial solution may be a Neural Form, written as:

$$\boldsymbol{\Psi_t(x, w) = A(x) + Z(x)N(x, w), \;\; x \in [a, b]}$$

where $\boldsymbol{A(x)}$ and $\boldsymbol{Z(x)}$ are functions **not containing parameters**[1].

$\boldsymbol{A(x)}$ satisfies by construction the BCs.
$\boldsymbol{Z(x)}$ vanishes **only** at the endpoints, and is constructed so that $\boldsymbol{\Psi_t(x, w)}$ satisfies the BCs as well.

[1] Unless asymptotic BCs are specified, e.g. $\Psi(x) \sim \exp(-\beta x^2)$

Boundary Conditions II

For the case: $\Psi(a) = y_1, \quad \Psi(b) = y_2$

$$A(x) = y_1 \frac{x-b}{a-b} + y_2 \frac{x-a}{b-a}, \quad Z(x) = (x-a)(x-b)$$

while for the case: $\Psi(a) = y_1, \quad \Psi'(b) = y_2'$

$$A(x) = y_1 + y_2'(x-a), \quad Z(x) = (x-a)(x-b)^2$$

For $x \in R$, (Ordinary DEs) the BCs are trivially accommodated.

For PDEs where $x \in R^n, \ n = 2, 3, \cdots$, with **Dirichlet** and **Neumann** conditions, constructing $A(x)$ and $Z(x)$ may be complicated.

For rectangular boundaries there is a systematic way of construction.

Boundary Conditions III: Constraint Alternatives

Let $\mathcal{D}$ be a second order differential operator. Then the equation:

$$\mathcal{D}\Psi(x) = f(x), \quad x \in [a, b]$$

with BCs: $\Psi(a) = y_1, \quad \Psi'(b) = y_2'$, may alternatively be solved as:

Set: $\Psi_t(x, w) = N(x, w)$, and minimize:

$$E_\lambda(w) = \frac{1}{N} \sum_{i=1}^{N} \left[\mathcal{D}N(x_i, w) - f(x_i)\right]^2 +$$

$$+\lambda_1 \left[N(a, w) - y_1\right]^2 + \lambda_2 \left[N'(b, w) - y_2'\right]^2$$

In other words, **a simpler trial solution** is chosen,

and the BCs are **approximately satisfied**, using penalty methods.

In many dimensions, when the boundary is not a rectangular box, but is defined by a number of points a_i, a way to satisfy the BCs exactly, is to add a correction as:

$$\Psi_t(x, w) = N(x, w) + \sum_i q_i \exp\left(-\lambda(|x - \mu a_i + h|^2)\right), \quad h \in R^n$$

and solve the linear system (for Dirichlet BCs: $\Psi_t(a_j, w) = y_j$) :

$$y_j - N(a_j, w) = \sum_i q_i \exp\left(-\lambda(|a_j - \mu a_i + h|^2)\right)$$

and similarly for Neumann BCs: $\hat{n}_j^T \nabla \Psi_t(a_j, w) = c_j$.

The constants λ, μ, h are chosen so that the linear system for the q_i is numerically well behaved.

For Diriclet BCs: $\mu = 1, h = 0$ and $\lambda = [\min_{i \neq j} |a_i - a_j|]^{-1}$

For Neumann BCs: $\mu = 0.95, h^T = (0.1, 0.1, ...)$ and λ as above.

Systems of ODEs

Let a system of first order ODEs be:

$$\frac{d}{dx}\Psi_i(x) = f_i(x, \Psi_1, \Psi_2, \cdots, \Psi_n,), \;\; i = 1, 2, \cdots, n$$

with $x \in [0, 1]$, and $\Psi_i(0) = b_i, \;\; i = 1, 2, \cdots, n$.

The trial solutions are taken so as to satisfy the condition at $x = 0$:

$$\Psi_{i_t}(x, w) = b_i + xN_i(x, w)$$

$$E(w) = \sum_{i=1}^{n} \sum_{x_k \in [0,1]} \left[\frac{d}{dx}\Psi_{i_t}(x_k) - f_i(x_k, \Psi_{1_t}, \Psi_{2_t}, \ldots) \right]^2$$

Partial Differential Equations: Rectangular Boundaries

Let $\mathcal{D}$ be a second order partial differential operator. For instance:

$$\mathcal{D} = \alpha(x)\nabla^2 + \beta^T(x)\nabla + \gamma(x), \;\; x \in B^n \subset R^n$$

where B^n is a rectangular hyper-box in n dimensions, i.e.

$$B^n = [a_1, b_1] \otimes [a_2, b_2] \otimes \cdots, [a_n, b_n]$$

PDEs of the form: $\mathcal{D}\Psi(x) = f(x)$, with $\Psi(x)_{|\Omega_1}$ and $\hat{n}^T\nabla\Psi(x)_{|\Omega_2}$ specified, where Ω_1 and Ω_2 denote complementary parts of the hyper-box boundary, appear quite frequently in science and engineering problems.

Example in two dimensions

$$\nabla^2 \Psi(x,y) = f(x,y), x \in [0,1], y \in [0,1],$$

with **Dirichlet conditions:** i.e.:

$\Psi(x,0), \Psi(x,1), \Psi(0,y), \Psi(1,y)$ specified.

$$\Psi_t(x,y,w) = A(x,y) + Z(x,y)N(x,y,w)$$

with $Z(x,y) = x(1-x)y(1-y)$, and

Already Complicated !

$$\begin{aligned} A(x,y) = & (1-x)\Psi(0,y) + x\Psi(1,y) \\ & + (1-y)\left\{\Psi(x,0) - \left[(1-x)\Psi(0,0) + x\Psi(1,0)\right]\right\} \\ & + y\left\{\Psi(x,1) - \left[(1-x)\Psi(0,1) + x\Psi(1,1)\right]\right\} \end{aligned}$$

Complexity Increases with Dimension

- In **two dimensions**, the box has four sides.
  If on each side either Dirichlet or Neumann conditions are specified, there are $2^4 = 16$ possible combinations.

- In **higher dimensions** $n \geq 3$ there are $2^{2n} = 4^n$ different combinations, and forming the $\boldsymbol{A}(\boldsymbol{x})$ function becomes even more complex.

- If **Neural Forms** can be formed that satisfy by construction the BCs, they are to **be preferred** over the constrained optimization approach. They need **fewer** training points and satisfy the BCs **exactly**.

Boundary Matches - New Developments

Let $\boldsymbol{P}_{\boldsymbol{a,b}}^{\boldsymbol{i,j}}(\boldsymbol{x},\boldsymbol{f})$ be a polynomial of minimal degree satisfying:

$$\frac{\partial^i P_{a,b}^{i,j}}{\partial x^i}(x,f)|_{x=a} = f^{(i)}(a) \equiv \frac{\partial^i f}{\partial x^i}(x)|_{x=a}$$
$$\frac{\partial^j P_{a,b}^{i,j}}{\partial x^j}(x,f)|_{x=b} = f^{(j)}(b) \equiv \frac{\partial^j f}{\partial x^j}(x)|_{x=b}$$

Let $\boldsymbol{\mathcal{L}}_{\boldsymbol{x|a,b}}^{\boldsymbol{i,j}}$ be an operator defined as:

$$\mathcal{L}_{x|a,b}^{i,j} f(x) = P_{a,b}^{i,j}(x,f)$$

The function: $(1-\mathcal{L}_{x|a,b}^{i,j})f(x)$, has both the i^{th} derivative at $x=a$ and the j^{th} derivative at $x=b$, equal to zero.

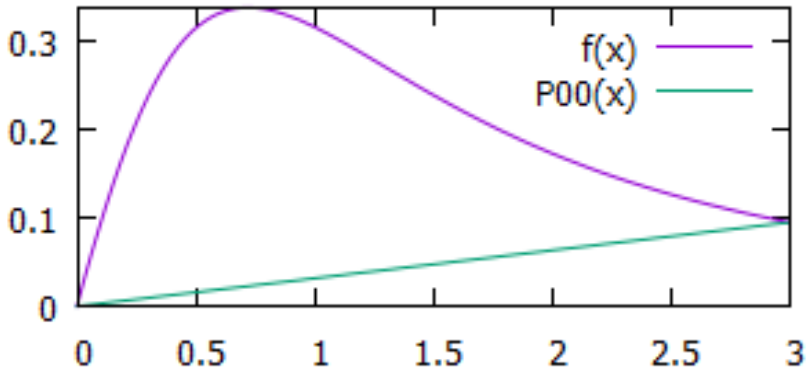
Boundary Matches - for $i,j \in \{0,1\}$

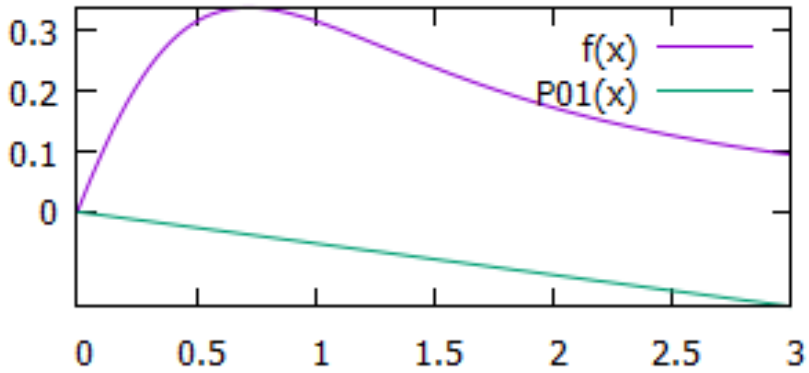
Some low–order common, cases:

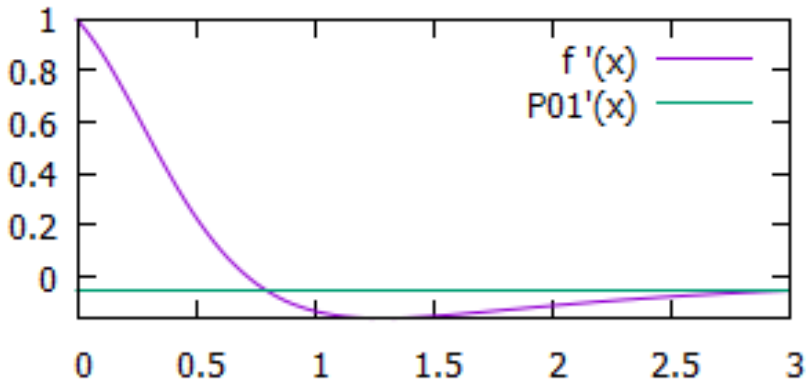
1. $P_{a,b}^{0,0}(x,f) = f(b)\dfrac{x-a}{b-a} - f(a)\dfrac{x-b}{b-a}$
2. $P_{a,b}^{0,1}(x,f) = (x-a)f^{(1)}(b) + f(a)$
3. $P_{a,b}^{1,0}(x,f) = (x-b)f^{(1)}(a) + f(b)$
4. $P_{a,b}^{1,1}(x,f) = \dfrac{1}{2}\dfrac{f^{(1)}(b) - f^{(1)}(a)}{b-a}x^2 + \dfrac{bf^{(1)}(a) - af^{(1)}(b)}{b-a}x$

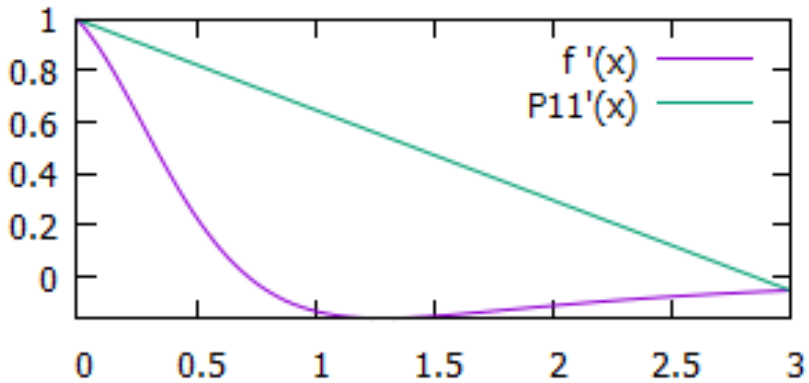
Boundary Matches

$$f(x) = \frac{1 - e^{-x}}{1 + x^2}, \quad f'(x) = \frac{df(x)}{dx} = \frac{e^{-x}}{1 + x^2} - \frac{2x(1 - e^{-x})}{(1 + x^2)^2}$$









Boundary Matches - General expressions

Case-I: $i < j$. $$P_{a,b}^{i,j}(x,f) = f^{(j)}(b)\frac{x^j}{j!} + \left[f^{(i)}(a) - f^{(j)}(b)\frac{a^{j-i}}{(j-i)!}\right]\frac{x^i}{i!}$$

Case-II: $i > j$. $$P_{a,b}^{i,j}(x,f) = f^{(i)}(a)\frac{x^i}{i!} + \left[f^{(j)}(b) - f^{(i)}(a)\frac{b^{i-j}}{(i-j)!}\right]\frac{x^j}{j!}$$

Case-III: $i = j$.

$$P_{a,b}^{i,i}(x,f) = \frac{f^{(i)}(b) - f^{(i)}(a)}{b-a}\frac{x^{i+1}}{(i+1)!} + \frac{bf^{(i)}(a) - af^{(i)}(b)}{b-a}\frac{x^i}{i!}$$

Boundary Matches, in two dimensions

Consider a function of two variables: $f(x,y),\ x \in [a,b],\ y \in [c,d]$.
The x-match operator acts (for example when $i = j = 0$) on $f(x,y)$ as:

$$\mathcal{L}^{0,0}_{x|a,b} f(x,y) = f(b,y)\frac{x-a}{b-a} - f(a,y)\frac{x-b}{b-a}$$

The y-match operator acts (for example when $i = 1, j = 0$) on $f(x,y)$ as:

$$\mathcal{L}^{1,0}_{y|c,d} f(x,y) = (y-d)\,\frac{\partial f}{\partial y}(x,c) + f(x,d).$$

The combined operation: (*Commutative)*

$$\mathcal{L}^{0,0}_{x|a,b}\,\mathcal{L}^{1,0}_{y|c,d} f(x,y) = \mathcal{L}^{1,0}_{y|c,d}\,\mathcal{L}^{0,0}_{x|a,b} f(x,y) =$$

$$\left[(y-d)\,\frac{\partial f}{\partial y}(b,c) + f(b,d)\right]\frac{x-a}{b-a} - \left[(y-d)\,\frac{\partial f}{\partial y}(a,c) + f(a,d)\right]\frac{x-b}{b-a}$$

Building the Boundary Match in two dimensions

The function:

$$A(x,y) = \left[1 - \left(1 - \mathcal{L}^{i,j}_{x|a,b}\right)\left(1 - \mathcal{L}^{k,m}_{y|c,d}\right)\right] f(x,y)$$
$$= \left(\mathcal{L}^{i,j}_{x|a,b} + \mathcal{L}^{k,m}_{y|c,d} - \mathcal{L}^{i,j}_{x|a,b}\mathcal{L}^{k,m}_{y|c,d}\right) f(x,y)$$

matches the following BCs:

$$\frac{\partial^i A}{\partial x^i}(a,y) = \frac{\partial^i f}{\partial x^i}(a,y), \quad \frac{\partial^j A}{\partial x^j}(b,y) = \frac{\partial^j f}{\partial x^j}(b,y)$$
$$\frac{\partial^k A}{\partial y^k}(x,c) = \frac{\partial^k f}{\partial y^k}(x,c), \quad \frac{\partial^m A}{\partial y^m}(x,d) = \frac{\partial^m f}{\partial y^m}(x,d)$$

Generalizing in many dimensions

Let, $x^T = (x_1, x_2, \cdots, x_N), \quad x_k \in [a_k, b_k]$.
Let, for x_k the BCs be represented by the match-operator $\mathcal{L}^{i_k, j_k}_{x_k | a_k, b_k}$.
Then the associated multidimensional boundary match is given by:

$$A(x) = \left[1 - \prod_{k=1}^{N} \left(1 - \mathcal{L}^{i_k, j_k}_{x_k | a_k, b_k} \right) \right] f(x)$$

and the corresponding Z-function is:

$$Z(x) = \prod_{k=1}^{N} (x_k - a_k)^{1+i_k} (x_k - b_k)^{1+j_k}$$

Non-linear system of ODEs[2]

$$\frac{\Psi_1(x)}{dx} = \cos(x) + \Psi_1^2(x) + \Psi_2(x) - (1 + x^2 + \sin^2(x))$$
$$\frac{\Psi_2(x)}{dx} = 2x - (1 + x^2)\sin(x) + \Psi_1(x)\Psi_2(x)$$
$$x \in [0, 3],\ \Psi_1(0) = 0,\ \Psi_2(0) = 1$$

with exact solution: $\Psi_1(x) = \sin(x), \Psi_2(x) = 1 + x^2$.

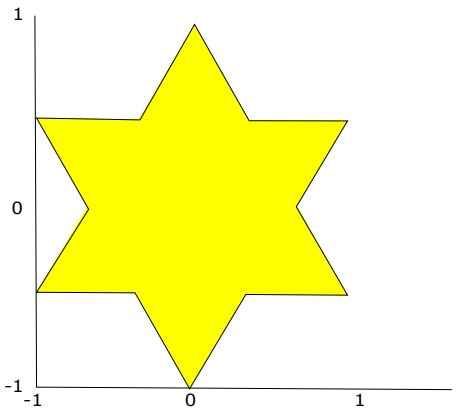
Trial solution: $\Psi_{1t} = xN(x, w_1)$ and $\Psi_{2t} = 1 + xN(x, w_2)$
Used 10 sigmoid hidden neurons for $N(x, w)$, 10 equidistant training points.
Achieved accuracy: 5 significant digits.

[2] *Artificial neural networks for solving ordinary and partial differential equations. IEEE Transactions on Neural Networks* **9** *(1998) 987*

Irregular Boundary non-linear PDE [3]

$$\nabla^2\Psi(x,y) + e^{\Psi(x,y)} = 1 + x^2 + y^2 + \frac{4}{\left(1+x^2+y^2\right)^2}$$

with the exact solution: $\Psi(x,y) = \log\left(1+x^2+y^2\right)$.



Solved for both Dirichlet and Neumann conditions inside the star-shaped domain, **treating the BCs as constraints with the Gaussian correction**.

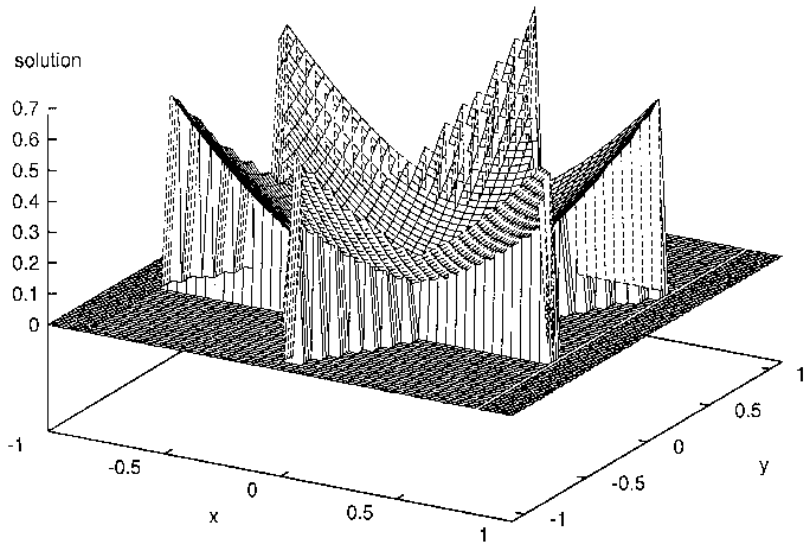
109 points were considered on the boundary, and **391** inside the star domain.

Using **20** hidden neurons, achieved **5** significant figures.

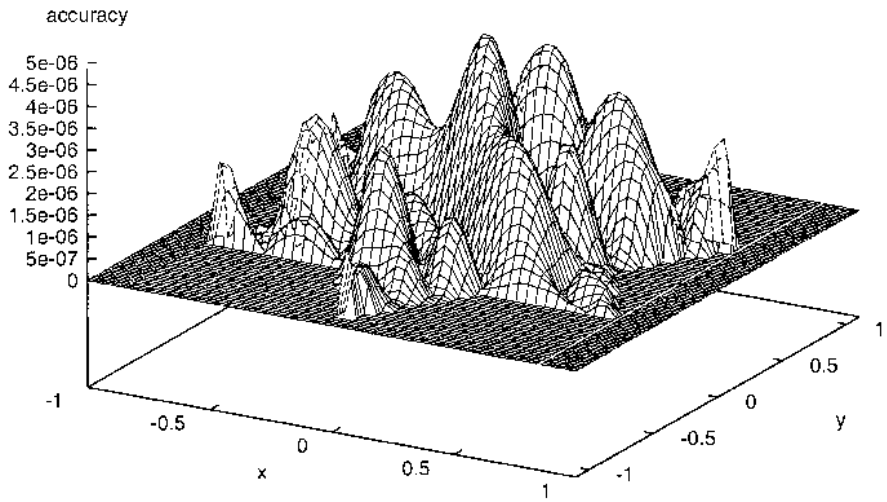
This problem has also been solved inside a cardioid.

[3] *Neural Network methods for boundary value problems with irregular boundaries. IEEE Transactions on Neural Networks* **11** *(2000) 1041*

Plots: Exact solution and absolute ”Error”



Plot of the exact solution



Asbolute deviation

Homogeneous DEs

They are Important - No need to Emphasize !!!

1. The Schrödinger equation: $-\nabla^2\Psi(x) + V(x)\Psi(x) = E\Psi(x)$
2. The Diffusion equation: $\dfrac{\partial u(x,t)}{\partial t} = \nabla \cdot (D(u,x)\nabla u(x,t))$
3. The Laplace equation: $\nabla^2\Phi(x) = 0$

The general approach is similar albeit with a few differences.

- The DE part (not the BCs) is satisfied by the trivial vanishing "solution".
- For vanishing BCs, the trivial solution may persist.
- In such cases, normalization usually settles the issue.

The Schrödinger Equation: $H\Psi = \mathcal{E}\Psi$

Let $\Psi_t(x, w)$ be the trial solution that respects the specified BCs.
$H\Psi_t(x, w) = \mathcal{E}\Psi_t(x, w)$, will be satisfied at a set of selected points x_i, by minimizing the relative "error":

$$E(w, \mathcal{E}) = \frac{\sum_i \left[H\Psi_t(x_i, w) - \mathcal{E}\Psi_t(x_i, w)\right]^2}{\sum_i \Psi_t(x_i, w)^2}$$

where $\mathcal{E} = \dfrac{\sum_i \Psi_t(x_i, w) H\Psi_t(x_i, w)}{\sum_i \Psi_t(x_i, w)^2}$, obtained by: $\dfrac{\partial}{\partial \mathcal{E}} E(w, \mathcal{E}) = 0.$

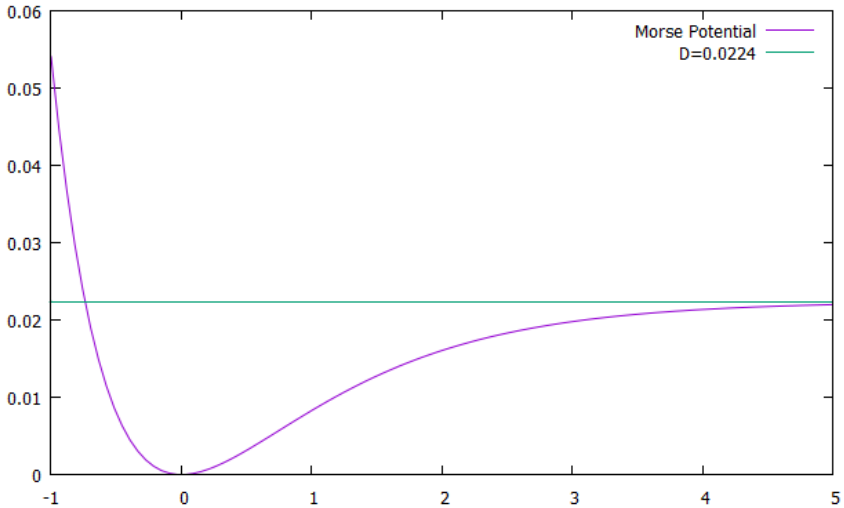
The denominator prevents convergence to the trivial solution.
Computationally, optimizing the relative error is clearly more complex.
Excited states may also be obtained, by a deflation approach.

The Schrödinger Equation: $H\Psi = \mathcal{E}\Psi$

Illustration via simple examples[4].

1-d Morse Potential: (For the I_2 molecule, $m = 119406$)

$V(x) = D[e^{-2ax} - 2e^{-ax} + 1]$ with $D = 0.0224, a = 0.9374$



$$\left[\frac{-1}{2m}\frac{d^2}{dx^2} + V(x)\right]\Psi(x) = \mathcal{E}\Psi(x).$$

For $x \to 0, V(x) \to Da^2x^2$

Harmonic oscillator.

$\Psi(0)$ finite, and $\Psi(x) \sim e^{-\beta x^2}$

The trial solution is written as: $\boldsymbol{\Psi_t(x) = \exp(-\beta x^2) N(x, w)}$

Exact $\mathcal{E}_0 = 0.286171979 \times 10^{-3}$, Calculated $\mathcal{E}_0 = 0.286171981 \times 10^{-3}$

Trained with 8 hidden nodes and 150 points $\in [-1, 2]$.

[4] *Artificial neural network methods in quantum mechanics. Comput. Phys. Commun. 104(1997)1-14*

Integrodifferential Schrödinger Equation:

The $\boldsymbol{n+\alpha}$ system, in the framework of the "Resonating Group Method", is described by the non-local Schrödinger Equation:

$$\left(\frac{-\hbar^2}{2m}\frac{d^2}{dr^2}+V(r)\right)\Psi(r)+\int_0^\infty K_0(r,r')\Psi(r')dr'=\mathcal{E}\Psi(r)$$

$$V(r)=-V_0\exp(-\beta r^2),\ V_0=41.28386,\ \beta=0.2751965$$
$$K_0(r,r')=-Ae^{-\gamma(r^2+r'^2)}(e^{2krr'}-e^{-2krr'})$$
$$(A=62.03772,\gamma=0.8025,\ k=0.46)$$

with $\Psi(0)=0$ and $\Psi(r)\sim e^{-\lambda r},\ \lambda>0$

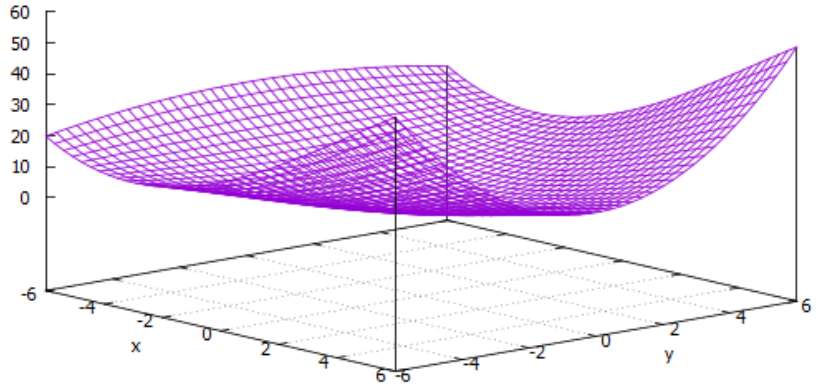
Trial solution: $\boldsymbol{\Psi_t(r)=r\exp(-\lambda r)N(r,w)}$

Trained with 8 hidden nodes and 100 points $\in[0,12]$.

Calculated $\mathcal{E}_0=-24.0764$ in agreement with other calculations.

2-d Henon-Heiles potential

$$V(x,y) = \frac{1}{2}(x^2+y^2) + \frac{xy^2 - x^3/3}{4\sqrt{5}}$$



$$(-\frac{1}{2}\nabla^2 + V(x,y))\Psi(x,y) = \mathcal{E}\Psi(x,y)$$

$\Psi(0,0)$ finite, $\Psi(x,y) \sim e^{-\beta(x^2+y^2)}$

Trial solution: $\boldsymbol{\Psi_t(x,y) = e^{-\beta(x^2+y^2)} N(x,y,w)}$, $\beta > 0$

Trained with 8 nodes and 20×20 points $\in [-6,6] \otimes [-6,6]$

Calculated $\mathcal{E}_0 = 0.99866$, in agreement with other calculations.

A Time-Saving Technique

It is well known that for the lowest eigenvalue $\mathcal{E}$ of H:

$$\mathcal{E} = \min_{\Psi} \frac{< \Psi | H | \Psi >}{< \Psi | \Psi >}$$

Minimizing $\dfrac{\sum_i \Psi_t(x_i, w) H \Psi_t(x_i, w)}{\sum_i \Psi_t(x_i, w)^2}$, leads to a fairly accurate estimation of $\mathcal{E}$.

However the resulting $\Psi_t(x, w)$ may not be as close to the ground state.

- Use this to estimate $\mathcal{E}$ and obtain an initial approximation to $\Psi_t(x, w)$.
- Then, minimize the relative error to obtain an accurate $\Psi_t(x, w)$ as well.

This will save a lot of unnecessary evaluations of the relative error, which is far more computationally demanding.

Diffusion Equation

$$\frac{\partial u(x,t)}{\partial t} = \frac{\partial}{\partial x}\left(D(u,x)\frac{\partial u(x,t)}{\partial x} \right), \quad x \in [0,1]$$

Initial Condition: $u(x,0)$ specified

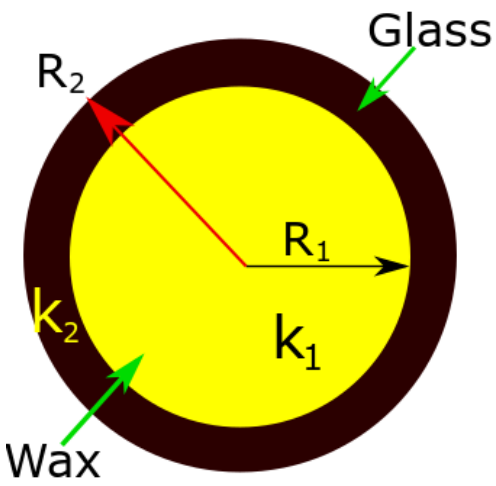
Dirichlet BCs: $u(0,t)$ and $u(1,t)$ specified

Trial solution: $u_t(x,t) = A(x,t) + x(1-x)(1-e^{-\lambda t})N(x,t,w)$

The factor $(1-e^{-\lambda t})$ accommodates the initial condition.
Note that λ must be positive to keep $u(x,t)|_{t\to\infty}$ finite.

$$\begin{aligned} A(x,t) &= \mathcal{L}^{00}_{x|01}u(x,t) + (1-\mathcal{L}^{00}_{x|01})u(x,0) \\ &= u(x,0) + (1-x)\left[u(0,t) - u(0,0)\right] + x\left[u(1,t) - u(1,0)\right] \end{aligned}$$

Heat Conduction



$$\rho c_p \frac{\partial T(r,t)}{\partial t} = k\left[\frac{\partial^2 T(r,t)}{\partial r^2} + \frac{1}{r}\frac{\partial T(r,t)}{\partial r}\right]$$

Initial Condition: $T(r,0)$ specified

BCs: $\dfrac{\partial T(0,t)}{\partial r} = 0,\ T(R_2,t)$ specified.

$$k = \begin{cases} k_1,\ r \in [0, R_1] \\ k_2,\ r \in (R_1, R_2] \end{cases}$$

At the glass-wax interface, $r = R_1$, the following conditions must hold:

$$\text{For } \epsilon \to 0,\ T(R_1 - \epsilon, t) = T(R_1 + \epsilon, t)$$

$$k_1 \frac{\partial T(R_1 - \epsilon, t)}{\partial r} = k_2 \frac{\partial T(R_1 + \epsilon, t)}{\partial r}$$

Important note: There is a **discontinuity** in $\frac{\partial T(r,t)}{\partial r}$ at $r = R_1$.

Introducing the discontinuity

$$A(r,t) = T(r,0) + T(R_2,t) - T(R_2,0)$$
$$Z(r,t) = r^2(R_2 - r)(1 - e^{-\lambda t}), \quad \lambda > 0$$
$$T_t(r,t) = A(r,t) + Z(r,t)\left[N(r,t) + \boldsymbol{a(t)}|\boldsymbol{r} - \boldsymbol{R_1}|\right]$$

Given that $T(r,0)$ satisfies the Interface Conditions, $a(t)$ is then determined so that $T(r,t)$ satisfies them as well, yielding:

$$a(t) = \frac{k_1 - k_2}{k_1 + k_2}\left[\frac{2R_2 - 3R_1}{R_1(R_2 - R_1)}N(R_1,t) + \frac{\partial N(R_1,t)}{\partial r}\right]$$

Pros and Cons

Obvious Pros	Obvious Cons
Closed form solution	Requires non-linear optimization [5]
Continuous, differentiable solution	Multiple local minima

Other Pros	Other Cons
Interpolation quality control	Time consuming
Data Set Extension at will	Global optimization ?
Economical representation [6]	—

[5] *Basis functions methods require only the solution of linear systems*

[6] *Basis functions methods require many terms to obtain similar accuracy*

Future Endeavors

- Activation functions may impact the performance of DE solving. Customization for different classes of DEs may be worthwhile.
- Different types of Neural Networks, may also offer advantages for certain classes of DEs.
- Deep Neural Networks have recently been applied to PDEs[7] in high dimensions, and to free boundary problems.
- ANNs with an infinite number of nodes is a new class of networks[8] quite promising for solving ODEs and PDEs.

[7] J. Sirignano, K. Spiliopoulos, *"DGM: A deep learning algorithm for solving partial differential equations", J. Comp. Phys. 375 (2018) 1339-1364*

[8] K. Blekas, I.E. Lagaris, *"Artificial neural networks with an infinite number of nodes", IOP Conf. Series: Journal of Physics: Conf. Series 915 (2017) 012006*

- A Legendre-type ANN has been applied τo ODEs [9].
- A Chebyshev-type ANN has been recently used to solve PDEs [10].
- Deep Neural Networks have been applied to solve PDEs with complex boundaries[11]

[9] S. Mall, S. Chakraverty, "*Application of Legendre Neural Network for solving ordinary differential equations", Applied Soft Computing 43 (2016) 347-356,*

[10] S. Mall, S. Chakraverty, "*Single Layer Chebyshev Neural Network Model for Solving Elliptic partial differential equations", Neural Process. Lett. 45 (2017)825–840*

[11] Jens Berg, Kaj Nyström, *"A unified deep artificial neural network approach to partial differential equations in complex geometries"*, Neurocomputing 317 (2018) 28-41