A new type of Neural Network is presented, with a single hidden layer and an infinite number of neurons. To render the transition to the continuum, a neuron density is introduced, the network weights become functions of a continuous variable, and the conventional sum is replaced by an integral.
Talk Structure

Why a new Neural Network
Infinite number of nodes
Functionally Weighted RBF
Posteriori Ascertainments

Numerical Experiments
Extrapolation Performance
Training Techniques
Solving ODEs & PDEs
Feed Forward Neural Networks

There is a plethora of Feed Forward Neural Networks that differ in:

**Architecture:**
- Shallow
- Deep

**Number of Nodes:** Few or Many

**Activation:**
- Sigmoid: \( \sigma(x) = \frac{1}{1 + \exp(-x)} \)
- \( \tanh(x) = 2\sigma(2x) - 1 \)

- Gaussian: \( G(x, \mu, \sigma) = e^{-\frac{1}{2} \left( \frac{x-\mu}{\sigma} \right)^2} \)

- Multiquadric: \( \sqrt{1 + x^2} \)

- Thin plate spline: \( x^2 \ln(x) \)

- Legendre, Chebychev, Bernstein, ...
Why a new Neural Network, and what is expected from it?

A new network in order to be competitive should offer:

- Higher Accuracy
- Parametric Economy
- Enhanced Interpolation Generalization
- Enhanced Extrapolation Generalization

"Parametric Economy" and "Generalization" are *intimately* correlated !!!
The Way to Higher Accuracy

It is proved\(^1\) that single hidden layer networks can approximate any function, to any desired degree of accuracy provided that sufficient number of neurons are available.

Hence, to obtain ultimate accuracy, the number of neurons should tend to $\textbf{Infinity}$.

\(^1\)K. Hornik, M. B. Stinchcombe, H. White, Neural Networks 2(1989)359-366, Multilayer feedforward networks are universal approximators
Gaussian RBF Networks

A Gaussian RBF Network with $K$ nodes (neurons), is given by:

$$N_G(x, \theta) = \sum_{i=1}^{K} A_i e^{-\frac{1}{2} \left( \frac{|x-\mu_i|}{\sigma_i} \right)^2} \equiv \sum_{i=1}^{K} A_i G(x, \mu_i, \sigma_i)$$

where $\theta$ stands collectively for all $\{A_i, \mu_i, \sigma_i\}$.

What happens when $K \rightarrow \infty$?
Disaster ... at First Sight

1. Number of parameters (weights): Infinite !!!
2. Computational Task: Impossible !!!
3. Approximation: Exact but Worthless !!!
4. Generalization: Infeasible !!!

With four parameters I can fit an Elephant, and with five I can make him wiggle his Trunk.

John von Neumann
Transition to the Continuum$^2$

In Physics this is a familiar limiting procedure ...

- The continuum limit of a chain, is a string.
- Discrete points are replaced by a point density.
- Differences become Derivatives.
- Indexed quantities become functions.
- Sums become Integrals.

---

Functionally Weighted Networks

**Standard RBF:** \( \mathcal{N}_G(x, \theta) = \sum_{i=1}^{K} A_i e^{-\frac{1}{2} \left( \frac{|x-\mu_i|}{\sigma_i} \right)^2} \)

**Introduce the neural node density:** \( \rho(s) \geq 0, \ s \in S \subset R \)

Such that: \( K = \int_S \rho(s) ds \to \infty \)

- \( A_i \to A(s) \)
- \( \mu_i \to \mu(s) \)
- \( \sigma_i \to \sigma(s) \)
- \( \sum_i \to \int_S ds \rho(s) \)

**FW-RBF:**

- \( \mathcal{N}_G(x, \theta) \to \mathcal{N}_{FW}(x, \theta) \equiv \int_S ds \rho(s) A(s) e^{-\frac{1}{2} \left( \frac{|x-\mu(s)|}{\sigma(s)} \right)^2} \)
Choices for $S$ and $\rho(s)$

Multitude of choices that satisfy: $\int_S ds \rho(s) \to \infty$, $\rho(s) \geq 0$

1. $S = (-\infty, \infty)$, $\rho(s) = 1$
2. $S = [0, 1]$, $\rho(s) = s^{-1}$
3. $S = [-1, 1]$, $\rho(s) = (1 - s^2)^{-1}$
4. ... ... ...

We have considered the third option: $\rho(s) = \frac{1}{1 - s^2}$, with $s \in [-1, 1]$

$$N_{FW}(x, \theta) \equiv \int_{-1}^{+1} \frac{ds}{1 - s^2} A(s) e^{-\frac{1}{2} \left( \frac{|x - \mu(s)|}{\sigma(s)} \right)^2}$$
A Technical Note
The Gauss-Chebyshev rule, known to be highly accurate, is given by:

\[
\int_{-1}^{+1} \frac{f(s)}{\sqrt{1 - s^2}} ds \approx \frac{\pi}{N} \sum_{i=1}^{N} f(s_i)
\]

where: \( s_i = \cos \left( \frac{2i - 1}{2N} \pi \right) \), \( \forall \ i = 1, 2, \cdots, N \)
Why a new Neural Network
The Way to Higher Accuracy
Gaussian RBF Networks
Infinite number of nodes
Functionally Weighted RBF

FW-RBF Final Form

Setting: \( w(s) \equiv \frac{A(s)}{\sqrt{1 - s^2}} \), the expression for the FW-RBF becomes:

\[
N_{FW}(x, \theta) = \int_{-1}^{+1} \frac{ds}{\sqrt{1 - s^2}} w(s) e^{-\frac{1}{2} \left( \frac{|x - \mu(s)|}{\sigma(s)} \right)^2}
\]

Remaining task is to choose the functions \( w(s), \mu(s), \sigma(s) \).
Let the data dimension be $d$. Then $\mu = (\mu_1, \cdots, \mu_d)^T \in \mathbb{R}^d$.

**Polynomial forms:**

\[
\begin{align*}
\bullet \ w(s) &= \sum_{i=0}^{L_w} w_i s^i \\
\bullet \ \mu_m(s) &= \sum_{i=0}^{L_\mu} \mu_{mi} s^i \\
\bullet \ \sigma(s) &= \sum_{i=0}^{L_\sigma} \sigma_i s^i
\end{align*}
\]

Total number of parameters: $L = L_w + d \times L_\mu + L_\sigma + d + 2$

**Ellipsoidal forms:**

\[
\mu_i(s) = u_i + v_i \frac{s + b_i}{\sqrt{\sum_{k=1}^{d} (s + b_k)^2}}, \quad w(s) \text{ and } \sigma(s) \text{ as above.}
\]

Total number of parameters: $L = L_w + L_\sigma + 3 \times d + 2$

**The number of adjustable parameters is certainly finite!!!**
Simple Cases

For $L_{\mu} = 1$ and $L_{\sigma} = 0$, 
$\mu(s) = \mu_0 + s\mu_1$ and $\sigma(s) = \sigma_0$

The locus of $\mu(s)$, the width $\sigma_0$, and the data points.
Posteriori Ascertainments

- Performed tests using data sets created by known functions.
- Each set was split in two subsets for Training and Testing.
- The training was performed both with and without “noise”.
- The testing subset remained clean (noise free).

Our Findings:

- Generalization in interpolating is superior.
- The generalization performance relative to other networks, increases with the noise level. (Noise Filter).
- FWNN is by far more economical compared to other networks.
- The generalization in extrapolating, clearly has an edge.
### Test functions in 1-d

(a) \( f(x) = 2x^2 + \exp(\pi/x) \sin(2\pi x) \)

(b) \( f(x) = x \sin(x) \cos(x) \)

(c) \( f(x) = \sin(x^2) - 0.25x \)

(d) \( f(x) = x \sin(x^2) \)
Why a new Neural Network
The Way to Higher Accuracy
Gaussian RBF Networks
Infinite number of nodes
Functionally Weighted RBF
Choices for $s$ and $\rho (s)$
Gauss-Chebyshev Quadrature
FW-RBF Final Form
Parametrize-Economize
Simple Case Illustration
Posteriori Ascertainments
Numerical Experiments
Extrapolation Performance
Training Techniques
Small Residual Problems
Solving ODEs & PDEs

FWNN configuration: $L_w = 5$, $L_\mu = 1$, $L_\sigma = 1$.  
Number of Parameters. **FWNN: 10**, MLP: 90/300, RBF: 30

**NMSE** stands for the “Normalized Mean Squared Error”:

$$E_{NMSE}(\theta) = \frac{1}{M} \sum_{i=1}^{M} \left( \frac{N(x_i, \theta) - f(x_i)}{\max(1, |f(x_i)|)} \right)^2 \times 100$$

<table>
<thead>
<tr>
<th>Method</th>
<th>NMSE over the TEST set</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>medium noise</td>
</tr>
<tr>
<td><strong>dataset 1(a)</strong></td>
<td></td>
</tr>
<tr>
<td>FWNN</td>
<td>0.63</td>
</tr>
<tr>
<td>MLP (best)</td>
<td><strong>0.59</strong> ($K = 30$)</td>
</tr>
<tr>
<td>RBF (best)</td>
<td>1.17 ($K = 10$)</td>
</tr>
<tr>
<td><strong>dataset 1(b)</strong></td>
<td></td>
</tr>
<tr>
<td>FWNN</td>
<td><strong>0.04</strong></td>
</tr>
<tr>
<td>MLP (best)</td>
<td>2.92 ($K = 100$)</td>
</tr>
<tr>
<td>RBF (best)</td>
<td>1.19 ($K = 10$)</td>
</tr>
</tbody>
</table>
Why a new Neural Network  
The Way to Higher Accuracy  
Gaussian RBF Networks  
Infinite number of nodes  
Functionally Weighted RBF  

**Gauss-Chebyshev Quadrature**  
**FW-RBF Final Form**  
**Parametrize-Economize**  
**Simple Case Illustration**  
**Posteriori Ascertainments**  
**Numerical Experiments**  
**Extrapolation Performance**  
**Training Techniques**  
**Small Residual Problems**  
**Solving ODEs & PDEs**

**FMSE over the TEST set**  
**medium noise**  
**high noise**

<table>
<thead>
<tr>
<th>Method</th>
<th>dataset 1(c)</th>
<th>dataset 1(d)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>medium noise</td>
<td>high noise</td>
</tr>
<tr>
<td><strong>FWNN</strong></td>
<td>0.03</td>
<td>0.24</td>
</tr>
<tr>
<td><strong>MLP</strong> (best)</td>
<td>3.67 (K = 30)</td>
<td>5.71 (K = 10)</td>
</tr>
<tr>
<td><strong>RBF</strong> (best)</td>
<td>3.83 (K = 20)</td>
<td>6.55 (K = 50)</td>
</tr>
<tr>
<td><strong>FWNN</strong></td>
<td>1.29</td>
<td>2.01</td>
</tr>
<tr>
<td><strong>MLP</strong> (best)</td>
<td>23.96 (K = 100)</td>
<td>48.19 (K = 100)</td>
</tr>
<tr>
<td><strong>RBF</strong> (best)</td>
<td>3.47 (K = 80)</td>
<td>5.77 (K = 80)</td>
</tr>
</tbody>
</table>

FWNN configuration: $L_w = 5$, $L_\mu = 1$, $L_\sigma = 1$.  
Number of Parameters. **FWNN: 10**, MLP: 90/300, RBF: 60/240
Test Functions in 2-d: Exponential and Gabor functions

(a) \( f(x_1, x_2) = x_1 \exp(- (x_1^2 + x_2^2)) \)

(b) \( f(x_1, x_2) = \frac{\pi}{2} \exp(-2(x_1^2 + x_2^2)) \cos(2\pi(x_1 + x_2)) \)
The Mexican Hat Function

\[ (c) \, f(x_1, x_2) = \frac{\sin(x_1^2 + x_2^2)}{\sqrt{x_1^2 + x_2^2}} \]

In each case 100 training and 1000 testing points were used.
### NMSE over the TEST set

<table>
<thead>
<tr>
<th>Method</th>
<th>medium noise</th>
<th>high noise</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>dataset 2(a)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FWNN</td>
<td>11.14</td>
<td>22.83</td>
</tr>
<tr>
<td>MLP (best)</td>
<td>19.84 ($K = 10$)</td>
<td>71.84 ($K = 10$)</td>
</tr>
<tr>
<td>RBF (best)</td>
<td>11.98 ($K = 50$)</td>
<td>51.73 ($K = 50$)</td>
</tr>
<tr>
<td><strong>dataset 2(b)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FWNN</td>
<td>1.55</td>
<td>4.66</td>
</tr>
<tr>
<td>MLP (best)</td>
<td>2.34 ($K = 100$)</td>
<td>7.95 ($K = 100$)</td>
</tr>
<tr>
<td>RBF (best)</td>
<td>1.69 ($K = 50$)</td>
<td>8.11 ($K = 30$)</td>
</tr>
<tr>
<td><strong>dataset 2(c)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FWNN</td>
<td>68.99</td>
<td>69.82</td>
</tr>
<tr>
<td>MLP (best)</td>
<td>84.97 ($K = 100$)</td>
<td>110.71 ($K = 100$)</td>
</tr>
<tr>
<td>RBF (best)</td>
<td>80.42 ($K = 80$)</td>
<td>86.18 ($K = 80$)</td>
</tr>
</tbody>
</table>

**Number of Parameters.** FWNN: **12**, MLP: 40/400, RBF: 120/200/320
Extrapolation Performance

Extrapolation is connected to prediction. Prediction is important!!!

*Make me a prophet,*

*and I will make you rich!!!*

- Does the FWNN extrapolate well?
- Is there a fair systematic comparison procedure?
- How does FWNN compare to the “competition”?
Comparison Setting

- Pick a test function $f(x)$.
- Choose 150 successive equidistant points.
- Train the networks (FWNN, MLP, RBF) using the first 100 points.
- Use the last 50 points: $x_1, \cdots, x_{50}$, for testing the extrapolation.

Extrapolation measure: $r_i \equiv \frac{|f(x_i) - N(x_i, \theta)|}{\max(1, |f(x_i)|)}$, the relative deviation.

For satisfactory extrapolation, $r_i$ should be small.

Let $d \in (0, 0.25]$ be an acceptable upper bound for $r_i$, i.e. $r_i \leq d$.

Determine $J$ such that: $r_i < d$, $\forall i \in [1, J]$ and $r_{J+1} \geq d$.

The network with the highest index $J$, is the extrapolation Winner.
Extrapolation Test Functions

\[ f_1(x) = x \sin(x) \cos(x) \]

\[ f_2(x) = \sin(x^2) - 0.25x \]
Extrapolation Performances

for: $f_1(x) = x \sin(x) \cos(x)$

for: $f_2(x) = \sin(x^2) - 0.25x$
## Extrapolation Comparison

<table>
<thead>
<tr>
<th>Network Architecture</th>
<th>Deviation bound: ( d )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.05</td>
</tr>
<tr>
<td>( f_1(x) = x \sin(x) \cos(x) )</td>
<td></td>
</tr>
<tr>
<td>( FWNN )</td>
<td>24 ± 3</td>
</tr>
<tr>
<td>MLP 10 nodes</td>
<td>12 ± 2</td>
</tr>
<tr>
<td>RBF 10 nodes</td>
<td>14 ± 2</td>
</tr>
<tr>
<td>( f_2(x) = \sin(x^2) - 0.25x )</td>
<td></td>
</tr>
<tr>
<td>( FWNN )</td>
<td>18 ± 1</td>
</tr>
<tr>
<td>MLP 10 nodes</td>
<td>6 ± 2</td>
</tr>
<tr>
<td>RBF 10 nodes</td>
<td>11 ± 1</td>
</tr>
</tbody>
</table>
Training Techniques

“Training” a Neural Network, is an optimization problem with the following “Sum-Of-Squares” objective function:

\[ E(\theta) = \sum_{i=1}^{M} [N(x_i, \theta) - y_i]^2 \equiv \sum_{i=1}^{M} [R(x_i, \theta)]^2 \]

Its gradient and Hessian given by:

\[ \nabla_\theta E(\theta) = 2 \sum_{i=1}^{M} R(x_i, \theta) \nabla_\theta R(x_i, \theta) \]

\[ \nabla^2_\theta E(\theta) = 2 \sum_{i=1}^{M} [\nabla_\theta R(x_i, \theta) \nabla_\theta R(x_i, \theta)^T + R(x_i, \theta) \nabla^2_\theta R(x_i, \theta)] \]
Small Residual Problems

If the model, i.e. the Network $N(x, \theta)$, is proper, then near the minimum point $\theta^*$, i.e. for $||\theta - \theta^*|| \leq \epsilon$, $R(x_i, \theta) \approx 0$.

This is called a “Small Residual Problem”, and in this case the Hessian may be approximated using first derivatives only as:

$$\nabla^2_\theta E(\theta) \approx 2 \sum_{i=1}^{M} \nabla_\theta R(x_i, \theta) \nabla_\theta R(x_i, \theta)^T$$

The indicated optimization methods therefore, belong to the so called “Gauss-Newton” class, either within the “Trust-Region” framework (Levenberg-Marquardt), or with the “line-search” approach.
Large Residual Problems

When for \( ||\theta - \theta^*|| \leq \epsilon, \ R(x_i, \theta) \gg 0 \), as for example in the case of “very” noisy data, we have a “Large Residual Problem”. In this case the “Gauss-Newton” approximation is not valid.

Appropriate methods are:

- “Modified Newton”
- “Quasi-Newton” (SR1, BFGS)
- “Limited Memory Quasi-Newton”
- “Conjugate Gradient” (Polak-Ribiere, Dixon, ... )
- “Hybrid Methods” (Fletcher & Xu\(^3\))

Solving ODEs & PDEs with FWNN

ANNs have been used in the past to solve ODEs and PDEs. A set of problems was considered and solved in a work\textsuperscript{4} entitled: “\textit{Artificial Neural Networks for solving ordinary and partial differential equations}”.

These problems have since been used as benchmarks by several authors who were developing methods for ODEs and/or PDEs, using various kinds and architectures of neural networks.

We have applied the same methodology using the FW-RBF Network instead of the MLP, on two of these problems (Problems #4 and #5).

\textsuperscript{4} \textit{IEEE Transactions on Neural Networks}, 9 (1998) 987-1000
System of ODEs. Problem #4

\[
\begin{align*}
\frac{d\Psi_1(x)}{dx} &= \cos(x) + \Psi_1^2(x) + \Psi_2(x) - (1 + x^2 + \sin^2(x)) \\
\frac{d\Psi_2(x)}{dx} &= 2x - (1 + x^2) \sin(x) + \Psi_1(x)\Psi_2(x) \\
\Psi_1(0) &= 0, \quad \Psi_2(0) = 1, \quad x \in [0, 3]
\end{align*}
\]

Exact solution: \( \Psi_1(x) = \sin(x), \quad \Psi_2(x) = 1 + x^2 \)

Trial Solution: \( \Psi_{1t}(x) = xN_1(x, \theta_1), \quad \Psi_{2t}(x) = 1 + xN_2(x, \theta_2) \)

Number of Points: 10 for Training and 100 for Testing.

**Preliminary Results**

<table>
<thead>
<tr>
<th>Network</th>
<th># of Parameters</th>
<th>MAD1</th>
<th>MAD2</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLP</td>
<td>30 ( = 10 \times 3)</td>
<td>2.0E-5</td>
<td>8.0E-5</td>
</tr>
<tr>
<td>FWNN</td>
<td>8 ( = 3 + 3 + 2)</td>
<td>5.0E-5</td>
<td>7.0E-5</td>
</tr>
</tbody>
</table>
PDE in 2-d. Problem #5

\[ \nabla^2 \Psi(x, y) = e^{-x}(x - 2 + y^3 + 6y), \quad (x, y) \in [0, 1] \otimes [0, 1] \]

Dirichlet BCs: \( \Psi(0, y) = y^3 \), \( \Psi(1, y) = \frac{1 + y^3}{e} \)

\[ \Psi(x, 0) = xe^{-x}, \quad \Psi(x, 1) = (x + 1)e^{-x} \]

with exact solution: \( \Psi(x, y) = e^{-x}(x + y^3) \).

Trial Solution: \( \Psi_t(x, y) = A(x, y) + x(1 - x)y(1 - y)N(x, y, \theta) \)

\[ A(x, y) = (1 - x)y^3 + x\frac{1 + y^3}{e} + (1 - y)x \left( e^{-x} - e^{-1} \right) \]

\[ + y \left[ (x + 1)e^{-x} - (1 - x + 2xe^{-1}) \right] \]
\[ \Delta \Psi(x, y) = \Psi(x, y) - \Psi_t(x, y) \]

Used a FWNN with 11 parameters \((3 + 2 \times 3 + 2)\)

A mesh of 100 points \((10 \times 10)\) was used for training, and a mesh of 900 points \((30 \times 30)\) for testing.

Absolute mean deviation \(\approx 1.3 \times 10^{-6}\).
Conclusions

The main features of the FWNN may be summarized as:

- "Economic" in the number of parameters.
- Excellent generalization while interpolating.
- Superior extrapolation capability.
Work to be done

- Only polynomials have been tried up to now for \( w(s), \mu(s), \sigma(s) \). More forms should be investigated.
- Sigmoidal FWNNs should also be explored, i.e.:
  \[
  N_{FW}^\sigma(x, \theta) = \int_{-1}^{1} \frac{ds}{\sqrt{1 - s^2}} a(s) \sigma(w^T(s)x + b(s))
  \]
- Extend FWNN to Deep-FWNN, to explore possible benefits.