Gradient-Enhanced Neural Network

Steven H. Berguin[*]

Georgia Institute of Technology, Atlanta, GA, 30332, USA

November 21, 2018

Nomenclature

a	activation output
b	bias (parameter to be learned)
g	activation function, $a = g(z)$
K	number of nodes in output layer
$\mathbf{J}$	Jacobian
L	number of layers in neural network
m	number of training examples
N	number of test examples
n	number of nodes in a layer
p	number of nodes in input layer
w	weight (parameter to be learned)
x	training data input
y	training data output
$\hat{y}$	predicted output
z	linear activation input

Hyper-Parameters

α	learning rate
γ	gradient enhancement parameter
λ	regularization parameter

Mathematical Symbols and Operators

$\mathcal{J}$	cost function
$\mathcal{L}$	loss function
$()'$	first derivative, $\partial()/\partial z$
$()_{j'}$	first derivative, $\partial()/\partial x_j$
$()_{j''}$	second derivative, $\partial^2()/\partial x_j^2$

Superscripts/Subscripts

i	i^{th} training example
j	j^{th} node of input layer
l	l^{th} layer of neural network
r	r^{th} node of *current* layer l
s	s^{th} node of *previous* layer $l-1$

[*]Research Engineer II, School of Aerospace Engineering

1 Overview

The gradient of a continuous function represents a rich source of information that can go a long way to improve statistical model predictions, while reducing the number of training examples needed to achieve the same predictive performance. Gradient-enhanced regression is nothing new (see for example Forrester et al. [1] for gradient-enhanced Kriging or Giannakoglou et al. [2] for gradient-enhanced neural networks), but the computational cost of computing the gradient for every training example is usually prohibitive for most engineering applications. However, there exist certain specialized applications such as computational fluid dynamics with the advent of adjoint methods, where gradient-enhanced regression becomes compelling. The purpose of this document is simply to explain the theory behind the GENN algorithm on GitHub [1] and offer some validation results for the sake of transparency.

2 Mathematical Derivation

This section summarizes the mathematical derivation behind gradient-enhanced neural networks. The main steps steps of any neural network algorithm are summarized in Fig. 1 and the theory involved with each one is treated separately in the following sections. Gradient-Enhanced Neural Networks (GENN) follow the same steps as standard Neural Networks (NN), except that the cost function used during parameter update is modified to account for partial derivatives and, consequently, so must the backpropgation step.

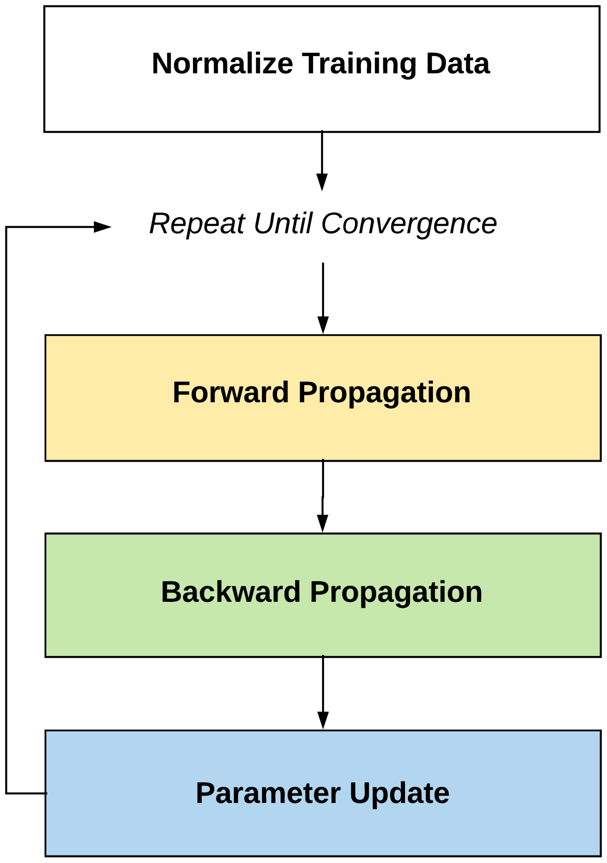

Figure 1: Summary of neural net training process

[1]git

2.1 Notation

For clarity, let us adopt the same notation as Andrew Ng [3] where superscript $[l]$ denotes the layer number, (i) is the training example, and subscript r is the node number of the current layer. For example, the activation of the r^{th} node in layer l for example i would be denoted:

$$a_r^{[l](i)} = g\left(z_r^{[l](i)}\right) \quad \forall \quad r \in [1, n^{[l]}] \quad i \in [1, m] \quad l \in [1, L] \tag{1}$$

To avoid clutter in the derivation to follow, it will be understood that subscripts r and s refer to quantities associated with the current and previous layer, respectively, without the need for superscripts. Concretely:

$$a_r \equiv a_{i=r}^{[l](i)} \quad \text{and} \quad z_r \equiv z_{i=r}^{[l](i)} \quad \forall \quad r \in [1, n^{[l]}] \tag{2}$$

$$a_s \equiv a_{i=s}^{[l-1](i)} \quad \text{and} \quad z_s \equiv z_{i=s}^{[l-1](i)} \quad \forall \quad s \in [1, n^{[l-1]}] \tag{3}$$

In turn, vector quantities will be defined in **bold** font. For instance, the activations associated with layer [l] would be given by:

$$\boldsymbol{a}^{[l](i)} = g\left(\boldsymbol{z}^{[l](i)}\right) \quad \text{where} \quad \boldsymbol{z}^{[l](i)} = \begin{bmatrix} z_1^{(i)} \\ \vdots \\ z_{n^{[l]}}^{(i)} \end{bmatrix}, \ \boldsymbol{a}^{[l](i)} = \begin{bmatrix} a_1^{(i)} \\ \vdots \\ a_{n^{[l]}}^{(i)} \end{bmatrix}, \tag{4}$$

This can be extended to matrix notation to account for all m training examples:

$$\mathbf{A}^{[l]} = \begin{bmatrix} a_1^{[l](1)} & \dots & a_1^{[l](m)} \\ \vdots & \ddots & \vdots \\ a_{n^{[l]}}^{[l](1)} & \dots & a_{n^{[l]}}^{[l](m)} \end{bmatrix} \quad \text{and} \quad \mathbf{Z}^{[l]} = \begin{bmatrix} z_1^{[l](1)} & \dots & z_1^{[l](m)} \\ \vdots & \ddots & \vdots \\ z_{n^{[l]}}^{[l](1)} & \dots & z_{n^{[l]}}^{[l](m)} \end{bmatrix} \tag{5}$$

$$\mathbf{A}_j'^{[l]} = \begin{bmatrix} a_1'^{[l](1)} & \dots & a_1'^{[l](m)} \\ \vdots & \ddots & \vdots \\ a_{n^{[l]}}'^{[l](1)} & \dots & a_{n^{[l]}}'^{[l](m)} \end{bmatrix} \quad \text{and} \quad \mathbf{Z}_j'^{[l]} = \begin{bmatrix} z_1'^{[l](1)} & \dots & z_1'^{[l](m)} \\ \vdots & \ddots & \vdots \\ z_{n^{[l]}}'^{[l](1)} & \dots & z_{n^{[l]}}'^{[l](m)} \end{bmatrix} \tag{6}$$

Finally, using the previous notation, the neural network parameters to be learned (*i.e.* weights and biases) are denoted as:

$$\mathbf{W}^{[l]} = \begin{bmatrix} w_{11}^{[l]} & \dots & w_{1n^{[l-1]}}^{[l]} \\ \vdots & \ddots & \vdots \\ w_{n^{[l]}1}^{[l]} & \dots & w_{n^{[l]}n^{[l-1]}}^{[l]} \end{bmatrix} \quad \text{and} \quad \boldsymbol{b}^{[l]} = \begin{bmatrix} b_1^{[l]} \\ \vdots \\ b_{n^{[l]}}^{[l]} \end{bmatrix} \quad \forall \quad l \in [1, L] \tag{7}$$

2.2 Training Data Normalization

The training data is given by $\mathbf{X}$ and $\boldsymbol{y}$ where there are $p = n^{[0]}$ inputs, $K = n^{[L]}$ outputs, and m training examples:

$$\mathbf{X}^{[l]} = \begin{bmatrix} x_1^{(1)} & \dots & x_1^{(m)} \\ \vdots & \ddots & \vdots \\ x_p^{(1)} & \dots & x_p^{(m)} \end{bmatrix} \quad \text{and} \quad \mathbf{Y}^{[l]} = \begin{bmatrix} y_1^{(1)} & \dots & y_1^{(m)} \\ \vdots & \ddots & \vdots \\ y_K^{(1)} & \dots & y_k^{(m)} \end{bmatrix} \tag{8}$$

In addition, gradient information might be available for each output. For a single training example, this can be stored as a Jacobian matrix:

$$\mathbf{J}^{(i)} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \dots & \frac{\partial y_1}{\partial x_p} \\ \vdots & \vdots & \vdots \\ \frac{\partial y_K}{\partial x_1} & \dots & \frac{\partial y_K}{\partial x_p} \end{bmatrix} \quad \forall \quad i \in [1, m] \tag{9}$$

For convenience, let $\boldsymbol{a}_j'^{(i)}$ be the j^{th} column of $\mathbf{J}^{(i)}$ such that:

$$\mathbf{A}_j' = \begin{bmatrix} \boldsymbol{a}_j'^{(1)} & \dots & \boldsymbol{a}_j'^{(m)} \end{bmatrix} = \begin{bmatrix} \frac{\partial y_1}{\partial x_j}^{(1)} & \dots & \frac{\partial y_1}{\partial x_j}^{(m)} \\ \vdots & \vdots & \vdots \\ \frac{\partial y_K}{\partial x_j}^{(1)} & \dots & \frac{\partial y_K}{\partial x_j}^{(m)} \end{bmatrix} \quad \forall \quad j \in [1, p] \tag{10}$$

Although normalizing the training data is optional, it can greatly improve performance when inputs (or outputs) differ by orders of magnitude. For example, in aerodynamics, one input might be the Reynold's number (measured on the order of millions) and another might be the angle of attack (measured in radians). A simple way to avoid numerical ill-conditioning is to subtract the mean and divide by the standard deviation of the data:

$$\boldsymbol{\mu}_x = \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{x}^{(i)} \quad \in \quad \mathbb{R}^p \tag{11}$$

$$\boldsymbol{\mu}_y = \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{y}^{(i)} \quad \in \quad \mathbb{R}^K \tag{12}$$

$$\boldsymbol{\sigma}_x = \sqrt{\frac{1}{m}\sum_{i=1}^{m} \left(\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_x\right)^2} \quad \in \quad \mathbb{R}^p \tag{13}$$

$$\boldsymbol{\sigma}_y = \sqrt{\frac{1}{m}\sum_{i=1}^{m} \left(\boldsymbol{y}^{(i)} - \boldsymbol{\mu}_y\right)^2} \quad \in \quad \mathbb{R}^K \tag{14}$$

The normalized data then becomes:

$$\boldsymbol{x}^{(i)} := \frac{\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_x}{\boldsymbol{\sigma}_x} \tag{15}$$

$$\boldsymbol{y}^{(i)} := \frac{\boldsymbol{y}^{(i)} - \boldsymbol{\mu}_y}{\boldsymbol{\sigma}_y} \tag{16}$$

$$\boldsymbol{a}_j'^{(i)} := \boldsymbol{a}_j'^{(i)} \times \frac{\boldsymbol{\sigma}_x}{\boldsymbol{\sigma}_y} \tag{17}$$

where is should be understood that vector operations are taken element-wise.

2.3 Forward Propagation

Predictions are obtained by propagating information forward throughout the neural network, starting with the input layer, in a recursive manner:

$$[a]_r^{n^{[l]}} = \sum_{s=1}^{n^{[l-1]}} g(z_r) \qquad \text{where} \quad z_r = w_{rs} a_s + b_r \qquad \forall\ l \in [1, L] \tag{18}$$

$$[a_j']_r^{n^{[l]}} = \sum_{s=1}^{n^{[l-1]}} g'(z_r) z_{jr}' \quad \text{where} \quad z_{jr}' = w_{rs} a_{js}' + b_r \quad \forall\ j \in [1, p] \tag{19}$$

Finally, the activation function is taken to be the hyperbolic tangent:

$$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \quad \Rightarrow \quad g'(z) \equiv \frac{\partial g}{\partial z} = 1 - g(z)^2 \tag{20}$$

Also, note that when $l = 0$ the activations are simply the input values, *i.e.*

$$\boldsymbol{a}^{[0]} = \boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_p \end{bmatrix} \quad \text{where} \quad p = n^{[0]} \tag{21}$$

Similarly, when $l = L$ the activations are simply the predicted output values:

$$\boldsymbol{a}^{[L]} = \hat{\boldsymbol{y}} = \begin{bmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_K \end{bmatrix} \quad \text{where} \quad K = n^{[L]} \tag{22}$$

2.4 Parameter Update

The neural network parameters are learned by solving the following, unconstrained optimization problem:

$$\theta = \arg\min_{\theta} \mathcal{J}(\theta) \quad \text{where} \quad \theta = \left\{ \mathbf{W}^{[l]}, \boldsymbol{b}^{[l]} \right\} \quad \forall \quad l \in [1, L] \tag{23}$$

The cost function $\mathcal{J}$ is given by the following expression, where λ is a hyper-parameter that controls regularization,

$$\mathcal{J}(\theta) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}^{(i)} + \frac{\lambda}{2m} \sum_{l=1}^{L} \sum_{r=1}^{n^{[l]}} \sum_{s=1}^{n^{[l-1]}} w_{rs}^2 \tag{24}$$

and the least squares loss function $\mathcal{L}$ accounts for partial derivatives,

$$\mathcal{L}^{(i)} = \frac{1}{2} \sum_{r=1}^{K} \left[\left(a_r^{[L](i)} - y_r^{(i)} \right)^2 + \gamma \sum_{j=1}^{p} \left(a_{jr}^{\prime[L](i)} - y_{jr}^{\prime(i)} \right)^2 \right] \tag{25}$$

where $a_r^{[L](i)} = \hat{y}_r(\theta, \boldsymbol{x})$ and γ is another hyper-parameter to be tuned. The parameters are updated iteratively for each layer using gradient-descent,

$$\theta := \theta - \alpha \frac{\partial \mathcal{J}}{\partial \theta} \tag{26}$$

where α is the learning rate.

2.5 Backward Propagation

Forward propagation seeks to compute the gradient of the predicted outputs $\hat{\boldsymbol{y}}$ with respect to (abbreviated w.r.t.) the inputs $\boldsymbol{x}$. By contrast, back-propagation computes the gradient of the cost function $\mathcal{J}$ with respect to the neural net parameters θ.

2.5.1 Cost Function Derivatives w.r.t. Neural Net Parameters

In a first step, assume the quantities $\partial \mathcal{L} / \partial a_r$ and $\partial \mathcal{L} / \partial a_r'$ are known and consider the gradient of the loss function $\mathcal{L}$ with respect to the parameters θ from layer l. Applying the chain rule to Eq. 25 yields:

$$\frac{\partial \mathcal{L}}{\partial \theta} = \frac{\partial \mathcal{L}}{\partial a_r} \frac{\partial a_r}{\partial z_r} \frac{\partial z_r}{\partial \theta} + \sum_{j=1}^{p} \frac{\partial \mathcal{L}}{\partial a_r'} \left(\frac{\partial a_{jr}'}{\partial z_r} \frac{\partial z_r}{\partial \theta} + \gamma \frac{\partial a_{jr}'}{\partial z_{jr}'} \frac{\partial z_{jr}'}{\partial \theta} \right) \tag{27}$$

where the various partial derivatives follow from Eqs. 18–19:

$$\frac{\partial a_r}{\partial z_r} = g'(z_r) \qquad \frac{\partial a_{jr}'}{\partial z_r} = g''(z_r) z_{jr}' \qquad \frac{\partial a_{jr}'}{\partial z_{jr}'} = g'(z_r) \tag{28}$$

$$\theta = w_{rs} : \quad \frac{\partial z_r}{\partial \theta} = a_s \qquad \frac{\partial z_{jr}'}{\partial \theta} = a_{js}' \tag{29}$$

$$\theta = b_r : \quad \frac{\partial z_r}{\partial \theta} = 1 \qquad \frac{\partial z_{jr}'}{\partial \theta} = 0 \tag{30}$$

Substituting the above in equation 27 yields:

$$\theta = w_{rs}: \quad \frac{\partial \mathcal{L}}{\partial w_{rs}} = \frac{\partial \mathcal{L}}{\partial a_r} g'(z_r) a_s + \sum_{j=1}^{p} \frac{\partial \mathcal{L}}{\partial a'_r} \left(g''(z_r) z'_{jr} a_s + g'(z_r) a'_{js} \right) \tag{31}$$

$$\theta = b_r: \quad \frac{\partial \mathcal{L}}{\partial b_r} = \frac{\partial \mathcal{L}}{\partial a_r} g'(z_r) + \gamma \sum_{j=1}^{p} \frac{\partial \mathcal{L}}{\partial a'_r} g''(z_r) z'_{jr} \tag{32}$$

Substituting into Eq. 24, the derivative of the cost w.r.t. parameters becomes:

$$\begin{aligned} \left[\frac{\partial \mathcal{J}}{\partial w_{rs}}\right]_{l=1}^{L} &= \lambda w_{rs} + \frac{1}{m} \sum_{i=1}^{m} \left[\frac{\partial \mathcal{L}^{(i)}}{\partial a_r} g'(z_r) a_s \right] \\ &\qquad + \gamma \sum_{i=1}^{m} \sum_{j=1}^{p} \frac{\partial \mathcal{L}^{(i)}}{\partial a'_r} \left(g''(z_r) z'_{jr} a_s + g'(z_r) a'_{js} \right) \end{aligned} \tag{33}$$

$$\left[\frac{\partial \mathcal{J}}{\partial b_r}\right]_{l=1}^{L} = \frac{1}{m} \sum_{i=1}^{m} \left[\frac{\partial \mathcal{L}^{(i)}}{\partial a_r} g'(z_r) + \frac{\gamma}{m} \sum_{j=1}^{p} \frac{\partial \mathcal{L}^{(i)}}{\partial a'_r} g''(z_r) z'_{jr} \right] \tag{34}$$

2.5.2 Loss Function Derivatives w.r.t. to Neural Net Activations

All quantities in Eqs. 33–34 are known, except for $\partial \mathcal{L}/\partial a_r^{[l]}$ and $\partial \mathcal{L}/\partial a'^{[l]}_{jr}$ which must be obtained recursively by propagating information backwards from one layer to the next, starting with the output layer L. The process is initialized from Eq. 25, where it follows from calculus that:

$$\frac{\partial \mathcal{L}}{\partial a_r^{[L]}} = \left(a_r^{[L]} - y_r \right) \quad \text{and} \quad \frac{\partial \mathcal{L}}{\partial a'^{[l]}_{jr}} = \left(a'^{[L]}_r - y'_r \right) \tag{35}$$

The derivatives for the next layer are obtained through the chain rule. According to Eq. 25, the loss function $\mathcal{L}$ depends on $n^{[l]}$ activations a_r and $p \times n^{[l]}$ activations a'_{jr}, which are all functions of a_s themselves. The same logic holds for $\partial \mathcal{L}/\partial a'_{jr}$. Hence, according to the chain rule:

$$\frac{\partial}{\partial a_s} \left(\mathcal{L}\Big(a_{r=1}, a'_{jr=1}, a_{r=2}, a'_{jr=2}, \ldots \Big) \right) = \sum_{r=1}^{n^{[l]}} \left[\frac{\partial \mathcal{L}}{\partial a_r} \frac{\partial a_r}{\partial a_s} + \gamma \sum_{j=1}^{p} \frac{\partial \mathcal{L}}{\partial a'_{js}} \frac{\partial a'_{js}}{\partial a_s} \right]$$

$$\frac{\partial}{\partial a'_{js}} \left(\mathcal{L}\Big(a_{r=1}, a'_{jr=1}, a_{r=2}, a'_{jr=2}, \ldots \Big) \right) = \sum_{r=1}^{n^{[l]}} \left[\frac{\partial \mathcal{L}}{\partial a_r} \frac{\partial a_r}{\partial a'_{js}} + \gamma \sum_{j=1}^{p} \frac{\partial \mathcal{L}}{\partial a'_{jr}} \frac{\partial a'_{jr}}{\partial a'_{js}} \right]$$

Further applying the chain rule to the intermediate quantities, then (using Eqs. 18–19) one finds that:

$$\begin{aligned} \frac{\partial a_r}{\partial a_s} &= \frac{\partial a_r}{\partial z_r} \frac{\partial z_r}{\partial a_s} \\ \frac{\partial a'_{jr}}{\partial a_s} &= \frac{\partial a'_{jr}}{\partial z_r} \frac{\partial z_r}{\partial a_s} + \frac{\partial a'_{jr}}{\partial z'_{jr}} \cancelto{0}{\frac{\partial z'_{jr}}{\partial a_s}} \\ \frac{\partial a_r}{\partial a'_{js}} &= 0 \\ \frac{\partial a'_{jr}}{\partial a'_{js}} &= \frac{\partial a'_{jr}}{\partial z_r} \cancelto{0}{\frac{\partial z_r}{\partial a'_{js}}} + \frac{\partial a'_{jr}}{\partial z'_{jr}} \frac{\partial z'_{jr}}{\partial a'_{js}} \end{aligned}$$

All expressions for the intermediate, partial derivatives above follow from Eqs. 18–19. Putting it all together, derivatives are obtained by working our way back recursively from layer $L, L-1, \ldots, 1$:

$$\begin{aligned}
\left[\frac{\partial \mathcal{L}}{\partial a_s}\right]^1_{l=L} &= \sum_{r=1}^{n^{[l]}} \left[\frac{\partial \mathcal{L}}{\partial a_r} \frac{\partial a_r}{\partial z_r} \frac{\partial z_r}{\partial a_s} + \gamma \sum_{j=1}^{p} \frac{\partial \mathcal{L}}{\partial a'_{jr}} \left(\frac{\partial a'_{jr}}{\partial z_r} \frac{\partial z_r}{\partial a_s} + \frac{\partial a'_{jr}}{\partial z'_{jr}} \cancelto{0}{\frac{\partial z'_{jr}}{\partial a_s}} \right) \right] \\
&= \sum_{r=1}^{n^{[l]}} \left[\frac{\partial \mathcal{L}}{\partial a_r} g'(z_r) w_{rs} + \gamma \sum_{j=1}^{p} \frac{\partial \mathcal{L}}{\partial a'_{jr}} g''(z_r) z'_{jr} w_{rs} \right] \qquad (36) \\
\left[\frac{\partial \mathcal{L}}{\partial a'_{js}}\right]^1_{l=L} &= \sum_{r=1}^{n^{[l]}} \left[\frac{\partial \mathcal{L}}{\partial a_r} \cancelto{0}{\frac{\partial a_r}{\partial a'_{js}}} + \gamma \sum_{j=1}^{p} \frac{\partial \mathcal{L}}{\partial a'_{jr}} \left(\frac{\partial a'_{jr}}{\partial z_r} \cancelto{0}{\frac{\partial z_r}{\partial a'_{js}}} + \frac{\partial a'_{jr}}{\partial z'_{jr}} \frac{\partial z'_{jr}}{\partial a'_{js}} \right) \right] \\
&= \sum_{r=1}^{n^{[l]}} \frac{\partial \mathcal{L}}{\partial a'_{jr}} g'(z_r) w_{rs} \qquad (37)
\end{aligned}$$

2.6 Key Equations in Vectorized Form

When dealing with large training data sets and large networks, vectorization can greatly help the efficiency of the training algorithm. To this end, key equations from the preceding material will be re-written in a vectorized form that can be implemented in Python, *i.e.* processing all m examples simultaneously (*i.e.* no loop).

2.6.1 Forward Propagation (Eqs. 18–19)

$$\mathbf{A}^{[l]} = g\left(\mathbf{Z}^{[l]}\right) \qquad \text{where} \quad \mathbf{Z}^{[l]} = \mathbf{W}^{[l]}\mathbf{A}^{[l-1]} + \boldsymbol{b}^{[l]} \quad \forall \quad l \in [1, L] \qquad (38)$$

$$\mathbf{A}_j^{'[l]} = g\left(\mathbf{Z}^{[l]}\right) \mathbf{Z}_j^{'[l]} \quad \text{where} \quad \mathbf{Z}^{'[l]} = \mathbf{W}^{[l]}\mathbf{A}_j^{'[l-1]} + \boldsymbol{b}^{[l]} \quad \forall \quad j \in [1, p] \qquad (39)$$

2.6.2 Parameter Update (Eq. 26)

$$\mathbf{W}^{[l]} := \mathbf{W}^{[l]} - \alpha \frac{\partial \mathcal{J}}{\partial \mathbf{W}^{[l]}} \quad \text{and} \quad \boldsymbol{b}^{[l]} := \boldsymbol{b}^{[l]} - \alpha \frac{\partial \mathcal{J}}{\partial \boldsymbol{b}^{[l]}} \quad \forall \quad l \in [1, L] \qquad (40)$$

$$\frac{\partial \mathcal{J}}{\partial \mathbf{W}^{[l]}} = \begin{bmatrix} \frac{\partial \mathcal{J}}{\partial w_{11}} & \cdots & \frac{\partial \mathcal{J}}{\partial w_{1n^{[l-1]}}} \\ \vdots & \vdots & \vdots \\ \frac{\partial \mathcal{J}}{\partial w_{n^{[l]}1}} & \cdots & \frac{\partial \mathcal{J}}{\partial w_{n^{[l]}n^{[l-1]}}} \end{bmatrix}, \quad \frac{\partial \mathcal{J}}{\partial \boldsymbol{b}^{[l]}} = \begin{bmatrix} \frac{\partial \mathcal{J}}{\partial b_1} \\ \vdots \\ \frac{\partial \mathcal{J}}{\partial b_{n^{[l]}}} \end{bmatrix} \quad \forall \quad l \in [1, L] \qquad (41)$$

2.6.3 Backward Propagation w.r.t. Parameters (Eqs. 33–34, Eqs. 36–37)

Let the operator $\circledast$ denote python broadcasting (not convolution) and the matrix operator $S_H(\cdot)$ denote the sum of each row.

$$\begin{aligned}
\frac{\partial \mathcal{J}}{\partial \mathbf{W}^{[l]}} = \frac{\lambda}{m}\mathbf{W}^{[l]} &+ \frac{1}{m}\sum_{k=1}^{K}\left(\frac{\partial \mathcal{L}}{\partial \mathbf{A}^{[l]}} \circledast g'(\mathbf{Z}^{[l]})\right)\mathbf{A}^{\top[l-1]} \\
&+ \frac{\gamma}{m}\sum_{k=1}^{K}\sum_{j=1}^{p}\left(\frac{\partial \mathcal{L}}{\partial \mathbf{A}_j^{'[l]}} \circledast g''(\mathbf{Z}^{[l]}) \circledast \mathbf{Z}_j^{'[l]}\right)\mathbf{A}^{\top[l-1]} \\
&+ \frac{\gamma}{m}\sum_{k=1}^{K}\sum_{j=1}^{p}\left(\frac{\partial \mathcal{L}}{\partial \mathbf{A}_j^{'[l]}} \circledast g'(\mathbf{Z})^{[l]}\right)\mathbf{A}_j^{'\top[l-1]} \qquad (42)
\end{aligned}$$

$$\frac{\partial \mathcal{J}}{\partial \boldsymbol{b}^{[l]}} = \frac{1}{m} S_H \left(\frac{\partial \mathcal{L}}{\partial \mathbf{A}^{[l]}} \circledast g'(\mathbf{Z}^{[l]}) + \sum_{j=1}^{p} \frac{\partial \mathcal{L}}{\partial \mathbf{A}_j'^{[l]}} \circledast g''(\mathbf{Z}^{[l]}) \circledast \mathbf{Z}_j'^{[l]} \right) \tag{43}$$

2.6.4 Backward Propagation w.r.t. Activations (Eqs. 36–37)

$$\frac{\partial \mathcal{L}}{\partial \mathbf{A}^{[l-1]}} = \mathbf{W}^{\top [l]} \left[\frac{\partial \mathcal{L}}{\partial \mathbf{A}^{[l]}} \circledast g'\left(\mathbf{Z}^{[l]}\right) + \gamma \sum_{j=1}^{p} g''\left(\mathbf{Z}^{[l]}\right) \circledast \mathbf{Z}_j^{[l]} \circledast \frac{\partial \mathcal{L}}{\partial \mathbf{A}_j'^{[l]}} \right] \tag{44}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{A}_j'^{[l-1]}} = \mathbf{W}^{\top [l]} \left[\gamma g'\left(\mathbf{Z}^{[l]}\right) \circledast \frac{\partial \mathcal{L}}{\partial \mathbf{A}_j'^{[l]}} \right] \tag{45}$$

3 Verification and Validation

In order to test that the GENN algorithm was properly implemented, the N-dimensional Rastrigin function was used:

$$f(\boldsymbol{x}) = \sum_{i=1}^{p} \left[x_i^2 - 10\cos(2\pi x_i) + 10 \right] \quad \forall \quad x_i \in [-1.0, 1.5] \tag{46}$$

$$\frac{\partial f}{\partial x_i} = 2x_i + 20\pi \sin(2\pi x_i) \tag{47}$$

The Rastrigin function is a popular test function for neural networks because it is easily extensible to many dimensions, highly multi-modal and, therefore, challenging to fit. The following section shows results for the 1D and 2D Rastrigin function because they can be easily visualized.

3.1 Test Function: 1D Rastrigin

In order to ensure an apples-to-apples comparison, the same hyper-parameters were used for both GENN and NN. In both cases, the ADAM optimization algorithm [4] was used for parameter updates:

$\alpha = 0.5$
$\gamma = 0$ (NN) or 1 (GENN)
$\lambda = 0.1$
$\beta_1 = 0.90$ (ADAM hyperparameter)
$\beta_2 = 0.99$ (ADAM hyperparameter)
hidden layer 1 has 24 nodes
hidden layer 2 has 12 nodes
hidden layer activation is tanh

Note that although only two hidden layers were used, the algorithm was written in a way that any number of layers could have been specified. The results for the one-dimensional Rastrigin function are shown in Fig. 2, where it can be seen that GENN outperforms NN and it only took 7 training examples to achieve an accurate prediction. This result suggests that the algorithm works and was properly implemented.

3.2 Test Function: 2D Rastrigin

The results for the 2D Rastrigin function are presented in Fig. 3, the same hyper-parameters as the 1D case were used. The dots shown in that figure correspond to training examples. Upon visual inspection, it can be seen that GENN outperforms NN once again, where the R-Square values were 0.998 and 0.915 in each case, respectively. Once again, results suggest valid implementation of the algorithm.

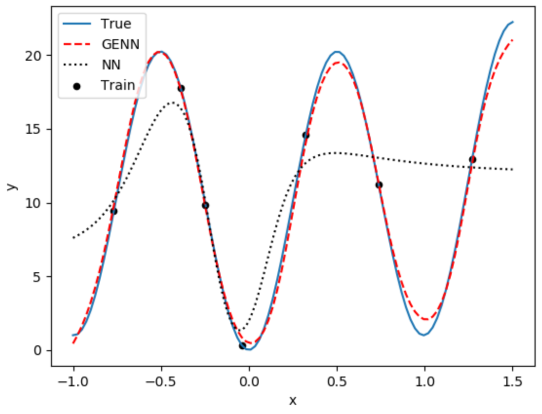

Figure 2: Comparison of GENN vs. NN using 1D Rastrigin

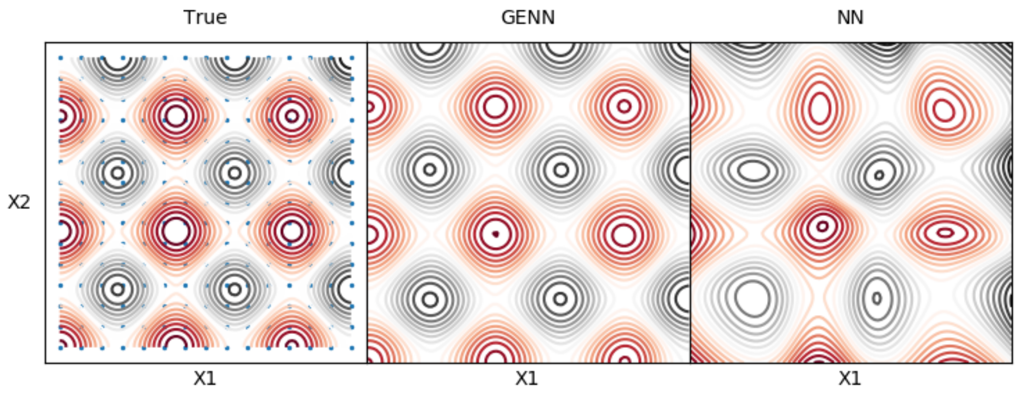

Figure 3: Comparison of GENN vs. NN using 2D Rastrigin

3.3 Test Function: ND Rastrigin

In order to assess the benefit of GENN in higher dimensions, a simple numerical experiment was constructed using the Rastrigin function as a benchmark. Specifically, GENN prediction accuracy was compared to NN for the different combinations of dimensionality p and sampling size m shown in Table 1, where the sampling density d is defined as the mean distance from the origin to the nearest data point **??**:

$$d(p,m) = \left(1 - \frac{1}{2}^{1/m}\right)^{1/p} \tag{48}$$

For each case, Halton sequences were used to ensure that points were uniformly distributed and all hyper-parameters were held fixed to the settings reported in the previous section. Finally, R-Square values were used to measure prediction accuracy over unseen test data:

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}{\sum_{i=1}^{N}(\hat{y}_i - \bar{y})^2} \tag{49}$$

While it is acknowledged that results are inherently problem-dependent, note that the Rastrigin function is particular relevant because it is smooth yet highly multi-modal, which makes it particularly challenging to regress without good coverage of the input space. It is therefore well-suited for this exercise.

Table 1: m as a function of p and d

p \ d	0.30	0.25	0.20	0.15	0.10
1	2	3	4	7	14
2	7	11	17	30	69
3	25	44	86	205	693
4	85	177	433	1,369	6,931

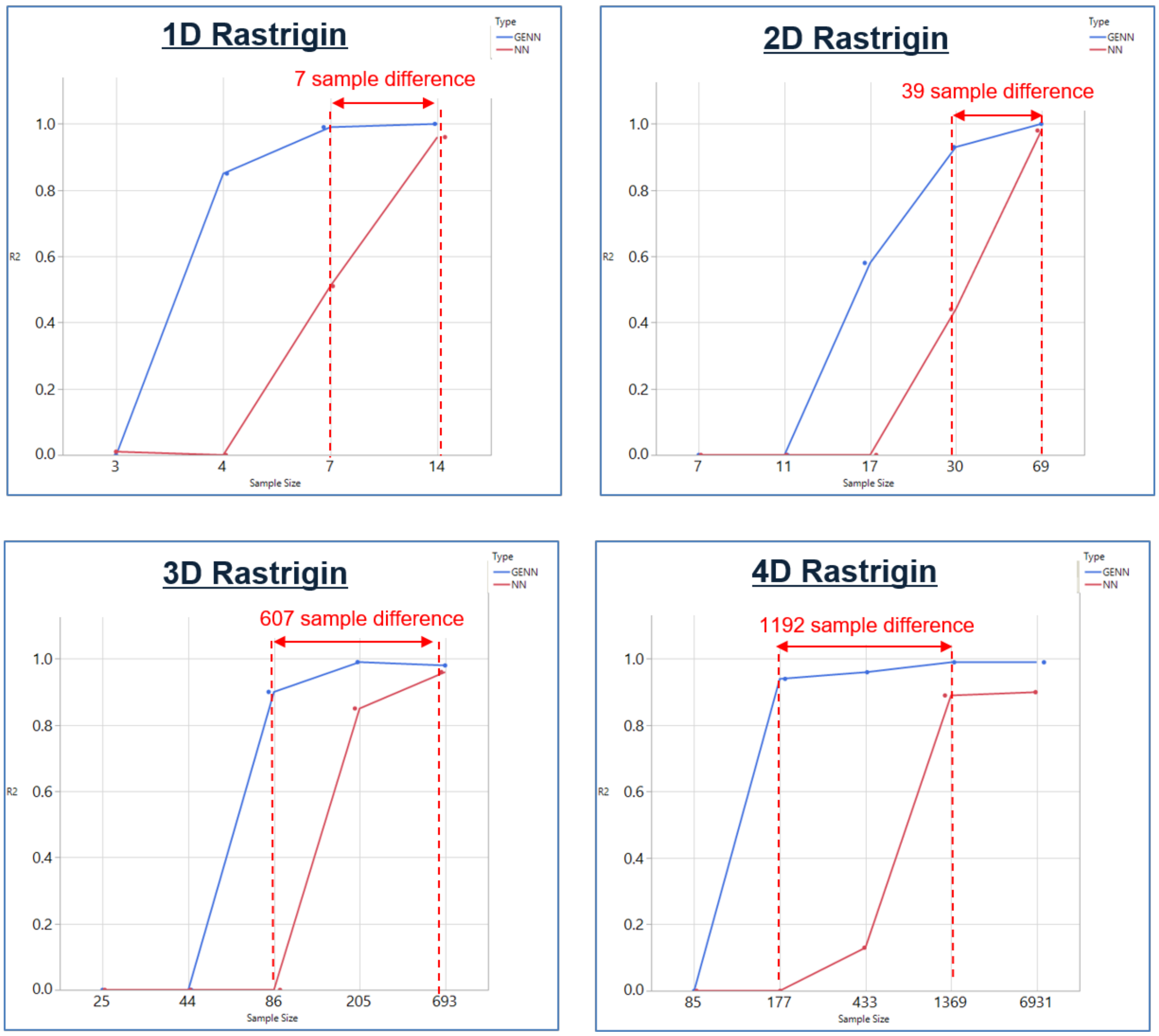

Figure 4: Comparison of GENN vs. NN as a function of sample size and dimensionality

Results are shown in Fig. 4, where it can be seen that the number of samples required to achieve the same R-Square is significantly lower for GENN compared to NN and this trend improves with dimensionality. Although this study did not consider dimensions higher than four, it is reasonable to expect that this trend carries into higher dimensions. Finally, for transparency, a typical goodness of fit for high R-Square values is shown in Fig. 5 and one for low R-Square is shown in Fig. 6.

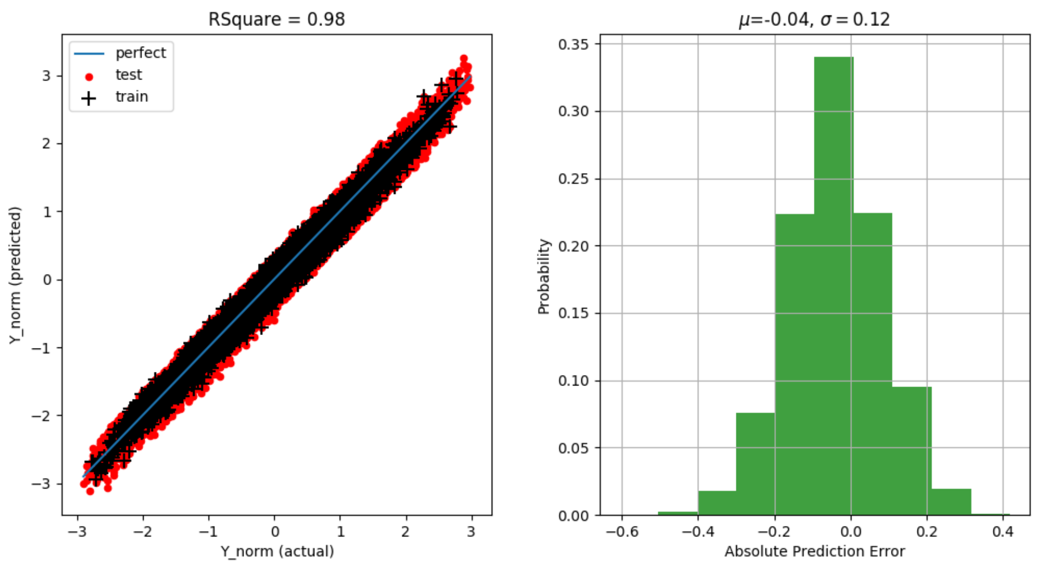

Figure 5: Example of a good fit – GENN, 4D Rastrigin with 6,931 samples. *The prediction error is normally distributed, centered near zero, and the standard deviation of the error is small compared to the range of variation of the response. Furthermore, both training and test data fall closely along the perfect fit line.*

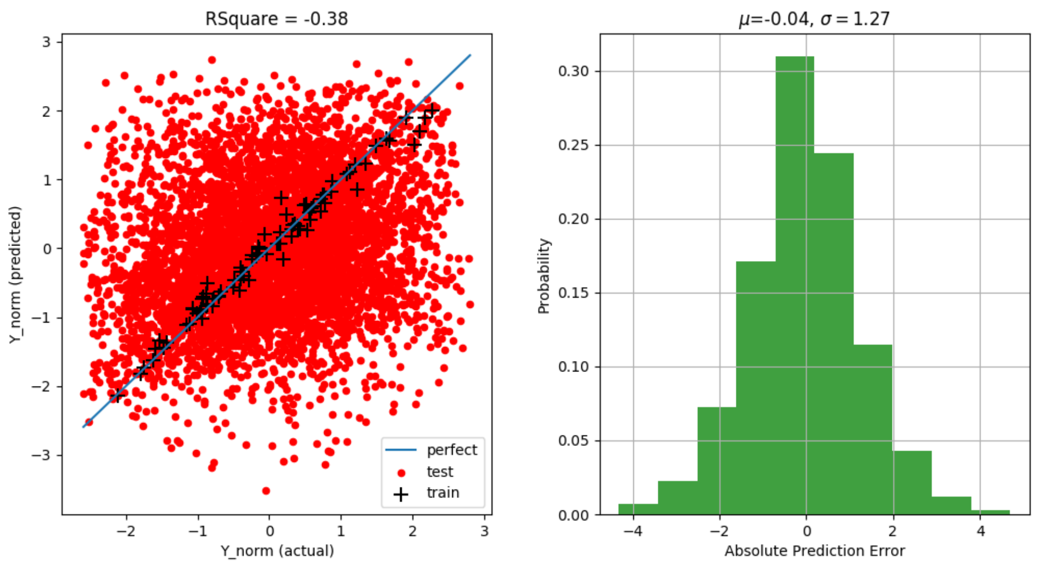

Figure 6: Example of a bad fit– NN, 3D Rastrigin with 86 samples. *The prediction error is normally distributed and centered near zero, but the standard deviation of the error is large compared to the range of variation of the response. Furthermore, the training falls nicely along the perfect fit line, but the test data is scattered all over the place, which is indicative of variance of error.*

References

[1] Forrester, A. I. J., Sóbester, A., and Keane, A. J., *Engineering Design via Surrogate Model*, John wiley & Sons, 2008.

[2] Giannakoglou, K. C., Papadimitriou, D. I., and Kampolis, I. C., “Aerodynamic shape design using evolutionary algorithms and new gradient-assisted metamodels,” *Computer Methods in Applied Mechanics and Engineering*, Vol. 195, No. 44-47, 2006, pp. 6312–6329.

[3] Andrew Ng, “Deep Learning — Coursera,” .

[4] Kingma, D. P. and Ba, J., “Adam: A Method for Stochastic Optimization,” 12 2014.