Numerical Condition of Feedforward Networks with Opposite Transfer Functions

Mario Ventresca and Hamid Reza Tizhoosh

***Abstract*— Numerical condition affects the learning speed and accuracy of most artificial neural network learning algorithms. In this paper, we examine the influence of opposite transfer functions on the conditioning of feedforward neural network architectures. The goal is not to discuss a new training algorithm nor error surface geometry, but rather to present characteristics of opposite transfer functions which can be useful for improving existing or to develop new algorithms. Our investigation examines two situations: (1) network initialization, and (2) early stages of the learning process. We provide theoretical motivation for the consideration of opposite transfer functions as a means to improve conditioning during these situations. These theoretical results are validated by experiments on a subset of common benchmark problems. Our results also reveal the potential for opposite transfer functions in other areas of, and related to neural networks.**

***Index Terms*— Numerical condition, ill-conditioning, opposite transfer functions, feedforward.**

I. Introduction

NUMERICAL condition is a very important and fundamental concept which affects the speed and accuracy of neural network learning algorithms [1]. Essentially, numerical condition refers to the sensitivity of the network output to changes in its weights and biases. If a network is ill-conditioned it may require long training times or converge to a poor solution.

Algorithms such as resilient propagation [2] quick propagation [3], conjugate gradient-based [4] and quasi-Newton-based [4] have the ability to adapt to an ill-conditioned situation. However, they will yield more desirable results when the network is well conditioned. Additionally, the condition will change during training if nonlinear hidden or output neurons are used [5]. Therefore, it is possible to improve conditioning both before or during (especially early stages) of training.

Some common approaches to help alleviate the detriments of ill-conditioning involve data preprocessing, weight initialization and regularization [6], [7], [4], [1]. Investigating the impact of transfer functions on the error surface has also been researched recently [8], [9], [10]. Also, adaptive transfer functions have been considered as a possible means to help alleviate ill-conditioning and improve accuracy and training times [11], [12], [13], [14].

M. Ventresca is a student member and H. R. Tizhoosh is a faculty member of the Pattern Analysis and Machine Intelligence (PAMI) laboratory in the Systems Design Engineering Department, University of Waterloo, Waterloo, ONT, N2L 3G1, CANADA (email: {mventres,tizhoosh}@pami.uwaterloo.ca)

This work has been supported in part by Natural Sciences and Engineering Council of Canada (NSERC).

In this paper we examine the concept of opposite transfer functions [15], [16], which represents a transformation of a neuron's transfer function such that it appears as a different location in weight space. The underlying motivation for this work lies in the area of Opposition-Based Computing, specifically Opposition-Based Learning (i.e. learning by considering opposite entities or actions) [17]. To date, opposition-based computing has led to improvements in reinforcement learning [18], [19], differential evolution [20], [21], [22], simulated annealing [23] and backpropagation [15], [16].

The remainder of this paper is organized as follows. Section II will discuss the concept and impact of symmetry on the output of neural network input-output mappings and introduce the idea of opposite transfer functions. Our theoretical motivations and proofs for the use of opposite transfer functions are provided in Section III. Experiments validating our theoretical results are provided in Section IV. Concluding results and directions for future work are presented in Section V.

II. Background

In this section we will provide an introduction to symmetries in weight space due to weight and transfer function transformations. Following this a description of opposite networks will be provided.

A. Preliminaries

In the following we consider, without loss of generality, a feedforward neural network $\mathcal{N}$ having $1 \leq i \leq n$ inputs, $1 \leq j \leq m$ hidden neurons and $1 \leq k \leq q$ output neurons. Input to the network is represented as a vector $x_p \in \mathbf{X}$, where $\mathbf{X}$ represents $p = 1...|\mathbf{X}|$ input patterns each of dimension n. The target output of x_p is denoted as $t_k(x)$ and corresponds to the $k = 1..q$ output neuron target value.

Using matrices $\mathbf{W} := (w_{i,j})_{n \times m}$ and $\mathbf{V} := (v_{j,k})_{m \times q}$ of input-to-hidden and hidden-to-output weight and bias values (for readability purposes we will also represent these as a single vector $\mathbf{Z} :=< w_{0,0}, ..., w_{n,m}, v_{0,0}, ..., v_{m,q} >$), we will refer to the input of each hidden neuron as

$$\psi_j(x) = \sum_{i=0}^{n} w_{i,j} x_i, \tag{1}$$

and the input to each output node as

$$\mu_k(x) = \sum_{j=0}^{m} v_{j,k} \varphi_j(\psi_j(x)), \tag{2}$$

where $\varphi_j(\psi_j(x))$ represents the output of hidden neuron j. The output of each terminal neuron is identified as $\phi_k(\mu_k(x))$, which we will refer to as $\phi_k(x)$. We assume, without loss of generality, that $\varphi_j(x)$ and $\phi_k(x)$ are the tanh and logistic functions, respectively. For ease in notation we define $\eta_a \in \mathcal{N}$ where $a = 1, ..., n, n+1, ..., n+m, n+m+1, ..., n+m+q$ to index any neuron in $\mathcal{N}$.

Letting $e_k(x) = t_k(x) - \phi_k(x)$ be the residual error, the mean squared error function

$$Er(\mathbf{X}) = \frac{1}{2|\mathbf{X}|} \sum_{x \in \mathbf{X}} \sum_{k=1}^{q} e_k^2(\mathbf{x}) \tag{3}$$

will be used to evaluate network performance.

B. Symmetry

Symmetry refers to a physical system's ability to remain unaffected by some transformation. In regards to neural networks this typically implies a transformation of the network parameters or structure which does not affect the input-output mapping represented by the network. That is, the input-output mapping $\Psi : \Re^n \mapsto \Re^q$ is invariant with respect to the transformation. Two networks $\mathcal{N}_1$ and $\mathcal{N}_2$ representing the same mapping are denoted as $\mathcal{N}_1 \sim \mathcal{N}_2$. In this work we concentrate on structural symmetry as it is concerned with transfer functions, but a more thorough examination can be found in [24].

Structural symmetries can be due to a permutation of neurons within a given layer or the sign inversion of a transfer function. For $\mathcal{N}$ having L hidden layers each of m_l neurons, we can permute any set of neurons from a specific layer by exchanging all input and output connections of the group of neurons. This permutation transformation does not affect the output of the network, and thus is a symmetrical transformation. In total, for m_l neurons there will be $m_l!$ equivalent weight vectors [25].

The other coherent transformation operates directly on the transfer function and is known as a sign transformation. Given a transfer function with odd symmetry (i.e. $\phi(x) = -\phi(-x)$), multiplying all input and output weights by -1 results in an invariant input-output mapping [26]. It has been shown that this specific symmetry is valid for any infinitely differentiable function where each successive differentiated function evaluates to zero [27].

If the transfer function exhibits even symmetry (i.e. $\phi(\cdot) = \phi(-\cdot)$) then multiplying all input connections by -1 also leaves $\Psi(\mathcal{N})$ unchanged. This symmetry is also valid for an infinitely differentiable function [27], of which the most common is the radial-basis transfer function. For either even or odd transfer functions, given a layer of m_l non-input neurons there exists 2^{m_l} possible sign-symmetric transformations.

The following remark summarizes the aforementioned symmetries [26]:

Remark The set of all equi-output transformations on the weight space Z forms a non-Albelian group G of order $\#G$, where

$$\#G = \prod_{l=2}^{L-1} (m_l!)(2^{m_l}) \tag{4}$$

where L is the number of non-input layers and m_l is the number of neurons in layer l.

Each of these transformations defines a symmetry in weight space consisting of equivalent parts. By taking these symmetries into consideration it is possible to reduce the size of the weight space [25], [26], [27], [28], [24], [29]. Further discussion on the influence of structural symmetry in neural networks can be found in [30]. Considering non-symmetric transformations may also be beneficial to neural network learning. Specifically, this paper is concerned with even-sign transformations on odd-signed transfer functions. The following subsection introduces this notion through the idea of opposite networks.

C. Opposite Networks

Before defining an opposite network we discuss the concept of an opposite transfer function (OTF). The underlying concept behind OTFs is to provide a means for altering the network structure such that knowledge stored in connection weights is retained but the input-output mapping differs. The purpose of each non-input neuron is to provide a "decision" or output based on the given signal. So, altering the neuron transfer function changes the corresponding decision rule of the neuron and consequently Ψ.

Dynamically adjusting transfer function parameters, and thus modifying the error surface and input-output mapping has been investigated, for example see [11], [12]. Similarly, many alternative transfer functions have been proposed, refer to [8] for a survey. However, most of these methods increase the search space size by (a) defining a set of transfer functions which can be used or, (b) increasing the parameterizations of the functions or, (c) infusing more parameters into the learning algorithm. As we will see below opposite transfer functions do imply that the size of search space increases.

An OTF, as used in this paper, is essentially an even transformation of an odd transfer function. An analogy for the idea is to view each neuron as a local decision maker which provides a decision (output signal) based on the given evidence (weighted input). Transforming the transfer function in a non-symmetric manner will then change the output signal of the neuron and possibly the network. An opposite transfer function can be defined as:

Definition *1 (Opposite Transfer Function):* Given some odd-symmetric transfer function $\varphi\colon \Re \mapsto \Re$, its corresponding opposite transfer function is $\breve{\varphi}(x) = \varphi(-x)$, where the breve notation indicates the function is an opposite.

This definition ensures that the relationship between a transfer function and its opposite is not odd (i.e. $\varphi(-x) \neq -\breve{\varphi}(x)$). It will be shown in Section III that OTFs also ensure that two irreducible networks differing in only the transfer function will yield different input-output mappings, that is $\mathcal{N}_1 \nsim \mathcal{N}_2$.

Another property of OTFs is that they do not lead to an increase in search space size. From definition 1 we notice that the transformation is equivalent to multiplying all input weights by -1, but not the output signal. This new weight configuration lies in the same weight space as the original network, and therefore there is no increase in search space size. So, the OTF is simply a means for examining the second location in weight space while retaining the knowledge stored in the unaffected network weights.

It should also be noted that in order to be useful in backpropagation-like learning algorithms the following characteristics should hold:

1) Both $\varphi(x)$ and $\breve{\varphi}(x)$ are continuous and differentiable.
2) For derivatives we have $\frac{d\breve{\varphi}(x)}{dx} = -\frac{d\varphi(x)}{dx}$.

Extrapolating on definition 1 we can now define the concept of an opposite network $\Gamma\mathcal{N}$.

Definition *2 (Opposite Network):* Given some minimal neural network $\mathcal{N}$ the corresponding set of opposite network(s) $\Gamma(\mathcal{N})$ is defined as all networks having identical connection structure and $\mathbf{Z}$ values, but differing in that at least one transfer function is in an opposite state.

Each $\gamma \in \Gamma(\mathcal{N})$ is simply a different point in weight space. We will show in the next section that $\gamma_a \nsim \gamma_b,\ \forall a \neq b$. Also, we show that without a-priori information each γ is equally likely to yield the most desirable performance before training begins.

III. Theoretical Results

In this section we provide theoretical foundation for our experimental results. We will show in a straightforward manner that considering opposite networks has advantages before and during early stages in training neural networks.

A. Network Irreducibility

Aside from symmetrical transformations, it is possible for $\mathcal{N}_1 \sim \mathcal{N}_2$ if one network can be reduced to the other [28]. For example, if there exists some neuron $\eta_a \in \mathcal{N}$ which has all outgoing weights equal to zero. Then, the removal of η_a does not affect Ψ. A formal definition of minimality, or equivalently, irreducibility has been given in [28]:

Definition *3 (Irreducibility):* A feedforward neural network with m input nodes and one hidden layer of n hidden neurons can be called irreducible if none of the following is true:

1) One of the $v_{j,k}$ vanishes.
2) There exists two indices $j_1, j_2 \in \{1, ..., n\}$ where $j_1 \neq j_2$ such that the functionals ψ_{j_1}, ψ_{j_2} are sign-equivalent[1].
3) One of the functionals ψ_j is constant.

An important consequence of minimality is that every minimal network represents a unique input-output mapping [28], [27].

[1]Two functions $f_1(x), f_2(x)$ are sign-equivalent if $f_1(x) = f_2(x)$ or $f_1(x) = -f_2(x)\ \forall x \in \Re^d$ where d is the dimensionality of the space.

We can now prove that each $\gamma \in \Gamma(\mathcal{N})$ represents a unique mapping if $\mathcal{N}$ is irreducible. The theorem trivially follows from the following three basic lemmas.

Lemma *1:* A vanishing $v_{j,k}$ only exists in some $\gamma_g \in \Gamma(\mathcal{N})$ if and only if it exists in $\mathcal{N}$.

Proof: Given that $\breve{\phi}(\mu_k(x)) = \phi(-\mu_k(x))$ is a symmetric transformation on μ_k then each $v_{j,k} = -v_{j,k}$ for the relevant neurons in γ_g. It follows trivially that the only instance where $-v_{j,k}$ vanishes is if $v_{j,k} = 0$. ■

Lemma *2:* Sign-equivalency can only exist in some $\gamma_g \in \Gamma(\mathcal{N})$ if and only if exists in $\mathcal{N}$.

Proof: Following a similar argument as in Lemma 1, an OTF maps $\psi_{j_1}(x) = -\psi_{j_1}(x)$. Sign-equivalency exists if $|\psi_{j_1}(x)| = |\psi_{j_2}(x)|$. So, substituting, we have $|-\psi_{j_1}(x)| = |\psi_{j_2}(x)|$. However, this can only occur if ψ_{j_1} and ψ_{j_2} were already sign-equivalent. ■

Lemma *3:* For $\gamma_g \in \Gamma(\mathcal{N})$, $\psi_j(x) = constant$ if and only if it is constant in $\mathcal{N}$.

Proof: This follows directly from Definition 1, which states that $\psi_j(x) = -\psi_j(x)$. ■

From these lemmas we now state the following theorem:

Theorem *1 (Opposite Network Irreducibility):* Given an irreducible network $\mathcal{N}$ all $\gamma_g \in \Gamma(\mathcal{N})$ are also minimal.

Proof: If every γ_g is minimal then it must obey the constraints outlined in definition 3. Each of the requirements is proven in Lemmas 1-3. ■

Let $S = \{\mathcal{N}\} \cup \Gamma(\mathcal{N})$, then from Theorem 1 every $s \in S$ represents a unique input-output mapping, denoted $\Psi(s)$.

Theorem *2 (Equi-Probable Input-Output Mapping):* Let $\mathcal{N}$ be a minimal neural network where $\mathbf{Z} \in \mathcal{U}(-\alpha, \alpha)$ (α, such that transfer functions avoid saturation) and with opposite networks $\Gamma(\mathcal{N})$. Without a-priori knowledge concerning $\mathbf{X}$ and for some $s^* \in S$,

$$P(s^* = \min(S)) = \frac{1}{|S|}$$

where,

$$|S| = \prod_{l \in L} 2^{m_l},$$

where L corresponds to the number of layers which can utilize opposite transfer functions, each having m_l neurons.

Proof: We make two common assumptions (the second makes the proof trivial):

1) The size and underlying distribution of $\mathbf{X}$ is unknown but is bounded by known finite bounds, which are scaled to [-1,1].
2) For $\mathcal{N}_1$ and $\mathcal{N}_2$ both minimal, having the same number of input, hidden and output neurons all using the same respective transfer functions, then

$$\begin{aligned} &P(Er_{\mathcal{N}_1}(\mathbf{X}) \leq Er_{\mathcal{N}_2}(\mathbf{X})) \\ =&P(Er_{\mathcal{N}_2}(\mathbf{X}) \leq Er_{\mathcal{N}_1}(\mathbf{X})) \\ =&0.5 \end{aligned}$$

From Theorem 1 and Definition 3 we know that

$$s_a \nsim s_b \ \forall s_a \neq s_b \in S.$$

Each transfer function can be either $\varphi(\cdot)$ or $\breve{\varphi}(\cdot)$, and so the total number of combinations is calculated by

$$|S| = \prod_{l \in L} 2_l^k.$$

Using Definition 1, each $s \in S$ represents a unique point in weight space such that

$$\mathbf{Z^a} \neq \mathbf{Z}^b, \forall s_a \neq s_b \in S,$$

where the superscript identifies which network the weight matrices belong to. From our assumptions each s is equally likely to yield the lowest error on $\mathbf{X}$ since they each exist within the same weight space. So,

$$P(s^* = \min(S)) = \frac{1}{|S|}.$$

■

While this holds for a random location in weight space, the rate at which this probability changes for each network during learning has not be determined analytically. We provide experimental evidence to support an exponential increase in probability.

B. Changes in the Jacobian and Hessian

The Jacobian matrix $\mathbf{J}$ is composed of first partial derivatives of the residual error for each pattern with respect to $\mathbf{Z}$. For a network with one output neuron $\mathbf{J}$ is computed as

$$\mathbf{J} = \left[\frac{\partial e(\mathbf{X})}{\partial z_i}\right]_{i=0\ldots,|Z|}. \tag{5}$$

Recall, that each $s \in S$ represents a unique mapping (i.e. $s_1 \nsim s_2$) implying their Jacobian matrices $\mathbf{J}(s)$ are different. So, for

$$\mathbf{\Delta}^J = \mathbf{J}(s_1) - \mathbf{J}(s_2) \tag{6}$$

it follows that $|\delta_{i,j}^J \neq 0| \geq 1$ for $\delta_{i,j}^j \in \mathbf{\Delta}^J$. Due to this property rank$(\mathbf{\Delta}^J) > 0$, where the rank of an $m \times n$ matrix $\mathbf{A}$ represents the number of linearly independent rows or columns. Rank deficiency occurs when rank$(\mathbf{A}) < \min(m, n)$.

Rank deficiency of the Jacobian is related to the concept of ill-conditioning [5], [31]. For backpropagation-like algorithms, a rank deficient Jacobian implies that only partial information of possible search directions is known, which can lead to longer training times. Furthermore, many optimization algorithms such as steepest decent, conjugate gradient, Newton, Gauss-Newton, Quasi-Newton and Levenberg-Marquardt directly utilize the Jacobian to determine search direction [5], [4].

The Hessian matrix $\mathbf{H}$ represents the second derivative of $Er(\mathbf{X})$ with respect $\mathbf{Z}$,

$$\mathbf{H} = \left[\frac{\partial^2 Er(\mathbf{X})}{\partial z_i z_j}\right]_{i,j=0\ldots,|Z|}. \tag{7}$$

The Hessian is very important to nonlinear optimization as it reveals the nature of error surface curvature. Specifically, the eigenvalues of $\mathbf{H}$ have a large impact on the learning dynamics of backpropagation-like algorithms. It is also employed by second-order learning algorithms [1], and the inverse $\mathbf{H}^{-1}$ can be used for network pruning strategies [32].

For neural networks it is common to compute

$$\mathbf{H} = \mathbf{J}^T\mathbf{J}. \tag{8}$$

As shown in Equation 6, utilizing some $\gamma_g \in \Gamma(\mathcal{N})$ will result in a change in $\mathbf{J}$. From (8) we should also expect a change in $\mathbf{H}$,

$$\mathbf{\Delta}^H = \mathbf{H}(s_1) - \mathbf{H}(s_2) \tag{9}$$

where there exists some $\delta_{i,j}^H \in \mathbf{\Delta}$ such that $\delta_{i,j}^H \neq 0$. Depending on the number and magnitude of the $\delta_{i,j}^H \neq 0$, the difference between the two positions in weight space could be significant enough to warrant moving the search to that location. This could be used as either a restart method or during learning.

The conditioning of $\mathbf{H}$ has a profound impact on the learning time and accuracy of the learning algorithm. The most common method to measure condition of $\mathbf{H}$ is through the condition number,

$$\kappa = \frac{\lambda_{max}}{\lambda_{min}} \tag{10}$$

where λ_{max} and λ_{min} are the largest and smallest nonzero eigenvalues of $\mathbf{H}$, respectively. The larger this ratio, the more ill-conditioned the network is.

IV. Experimental Results

In this section we provide results for two main experiments. Firstly, we examine the $Er(\mathbf{X})$, rank$(\mathbf{J})$ and κ at random points in the error surface for all combinations of transfer functions. The second experiment is aimed to examine the changes in the same three measures during early learning of a conjugate gradient algorithm.

A. Experimental Setup

Unless otherwise noted the following experiments all use a single hidden layer feedforward architecture with tanh and logistic transfer functions for the hidden and output neurons, respectively. Additionally, initial weights and biases are uniformly generated over $[-1, 1]$.

To evaluate the networks we utilized 3 common benchmark problems from the UCI-ML database [33] and two versions of the parity problem. These data sets, along with the number of hidden layers in the networks are presented in Table I. To avoid errors in Jacobian and Hessian calculation we standardize and normalize all input values. We also

remove records with missing data (very small percentage of the entire data set), although it is usually beneficial to keep these records, however, in this paper we are not concerned with the problem of handling missing data. Output values are binary and do not require adjustment.

TABLE I
THE BENCHMARK DATA AND THE NUMBER OF HIDDEN LAYERS USED.

Dataset	$\|\mathbf{X}\|$	Inputs	Hidden Size	$\|\mathbf{Z}\|$
3-bit parity	8	3	3	16
6-bit parity	64	6	6	49
Pima diabetes	768	8	5	51
Wisconsin Breast cancer	783[2]	9	5	56
ionosphere	351	33	5	176

The first experiment begins by generating a single neural network with random weights. Then, keeping $\mathbf{Z}$ constant, we evaluate $Er(\mathbf{X})$, rank($\mathbf{J}$) and κ for each $s \in S$. The process repeats for 4000 randomly generated $\mathcal{N}$.

Our second experiment focuses on the early stages of learning. Similar to the first experiment, we generate a random network $\mathcal{N}$. However, in this case we then train the network using Fletcher-Reeves Conjugate Gradient [4] for 10 epochs using scale factors $\alpha = 0.001$ and $\beta = 0.01$. At each epoch we record the $Er(\mathbf{X})$, rank($\mathbf{J}$) and κ for each $s \in S$, although we only train with respect to $\mathcal{N}$. In this manner, we can show the relationship between opposite networks and better understand how learning could benefit from considering opposite networks.

In both experiments we employ a minimum threshold value when calculating the rank and condition number. By doing this we ignore very small values which may skew our calculations. We only consider singular values of the Jacobian greater than 0.02, and eigenvalues of the Hessian greater than 0.02.

B. Before Training

The first experiment is aimed at comparing the conditioning of every transfer function combination before training begins. For m hidden nodes we have 2^m combinations where each combination represents a m-bit mask. For example, for $m = 2$ we have 4 combinations, $C = \{(00), (01), (10), (11)\}$, where a 0 or a 1 indicate whether the opposite transfer function is "off" or "on", respectively.

Figure 1 shows the result of random sampling using four of the benchmark data sets. As mentioned in the previous subsection, we generated 4000 random locations in weight space where we examined all 2^m combinations of transfer function. After 4000 samples these results provide experimental evidence to support Theorem 2.

Let $z_i \in Z$, $i = 1...4000$ be the i^{th} randomly sampled location then,

$$\mu = \frac{1}{4000} \sum_{i=1}^{4000} \min(Er_{S_{z_i}}(\mathbf{X})) \tag{11}$$

and σ is its standard deviation. In Table II we present μ and σ of the 4000 random samples. Each value is similar, but this reveals little about the actual bounds of initial network performance, Er. So we compute,

$$min = \min(\min(Er_{S_{z_i}}(\mathbf{X}))) \tag{12}$$

and,

$$max = \max(\min(Er_{S_{z_i}}(\mathbf{X}))) \tag{13}$$

to represent the lower and upper bound defined by the best $s \in S$ at each z_i. Using these measures we gain a better understanding of how the initial Er can vary than from simply examining σ. So, considering every $s \in S$ the Er range for each of the five benchmark problems is relatively large compared to $\mu \pm \sigma$. This is very representative of a network that is prone to ill-conditioning.

The final two comparison measures μ_{diff} and σ_{diff} represent the mean and standard deviation of $\max(Er_{S_{z_i}}(\mathbf{X})) - \min(Er_{S_{z_i}}(\mathbf{X}))$. Using these values, and computing μ_{diff}/μ for each problem we find the ratios 0.28, 0.26, 2.36, 0.65 and 1.05, respectively. Therefore, simply considering OTFs during network initialization can have a relatively influential impact on the initial network error.

TABLE II
A COMPARISON OF $Er(\mathbf{X})$ FOR EACH PROBLEM.

Dataset	μ	σ	min	max	μ_{diff}	σ_{diff}
3-Bit	0.25	0.02	0.20	0.31	0.07	0.04
6-Bit	0.27	0.02	0.23	0.36	0.07	0.02
cancer	0.14	0.04	0.07	0.33	0.33	0.11
diabetes	0.23	0.03	0.19	0.35	0.15	0.05
ionosphere	0.21	0.03	0.14	0.34	0.22	0.07

Using the methodology and measures described above, the rank($\mathbf{J}$) is explored in Table III. Additionally, all five problems have μ within 88.0% of the respective maximum rank, and so $\mathbf{J}$ will tend be rank deficient but not to a high degree. Comparing μ_{diff}/μ for each problem yields 0.03, 0.06, 0.01, 0.01 and 0.04, respectively. Thus, given a location in weight space the difference between the network with the highest and lowest rank($\mathbf{J}$) will be approximately 7% with respect to the mean rank for that specific problem.

TABLE III
A COMPARISON OF RANK($\mathbf{J}$) FOR EACH PROBLEM.

Dataset	μ	σ	min	max	μ_{diff}	σ_{diff}
3-Bit	7.6	0.57	4.00	8.00	0.24	0.43
6-Bit	41.71	2.93	21.00	47.00	2.56	1.23
cancer	54.25	3.36	34.00	56.00	0.49	1.00
diabetes	50.04	2.37	33.00	51.00	0.23	0.64
ionosphere	157.80	13.21	79.00	176.00	6.05	3.64

Table IV presents the results when comparing the impact of OTFs on the condition κ of $\mathbf{H}$. The key comparison is the ratio μ_{diff}/μ for each of the five problems, resulting in values: 1.24, 0.85, 1.24, 0.97 and 0.88, respectively. These values represent significant differences between the conditions of each $s \in S$ at a given weight configuration and further highlight the impact of OTFs before learning begins.

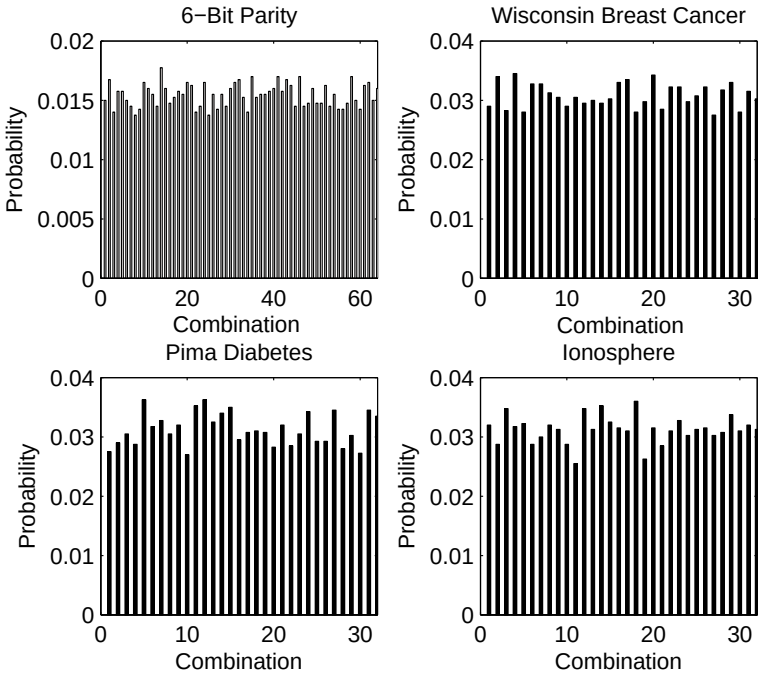

Fig. 1. Random sampling results for four benchmark problems. The probability of each combination yielding the lowest error is approximately uniform for each problem.

TABLE IV

A COMPARISON OF κ FOR EACH PROBLEM.

Dataset	μ	σ	min	max	μ_{diff}	σ_{diff}
3-Bit	9.0	4.5	2.1	27.5	11.2	7.2
6-Bit	146.9	21.6	46.3	264.9	125.0	32.8
cancer	3061.5	1030.1	869.9	8568.1	3798.0	1346.9
diabetes	1753.1	441.0	527.7	3979.0	1695.6	631.0
ionosphere	2284.3	687.2	846.6	6047.4	2004.1	978.4

C. During Early Training

This subsection provides insight into the impact of a learning algorithm on the usefulness of OTFs during the first 50 epochs of learning.

In Figure 2 we examine the probability a specific opposite network will yield the lowest $Er(\mathbf{X})$ assuming that it is solely trained a network with no opposite transfer functions. For this comparison we only consider the 3-bit parity problem because $|S|$ is small and thus the plot is more readable. After the second epoch 3 of the 8 networks show a non-zero probability of yielding the lowest error. By the fourth epoch the probability $\mathcal{N}$ has the lowest error is 1.0, keeping in mind the simplicity of the problem. The main purpose of this graph is to show an example of the behavior for each opposite network during training.

Figure 3 presents the probability that $\mathcal{N}$ with transfer function combination $(00...0)$ yields the lowest error when compared to $\Gamma(\mathcal{N})$. Except for the 3-bit parity problem, all probabilities are between 0.80 and 0.95 by the tenth epoch, where the behavior of probability increase is similar for each problem. Therefore, considering opposite networks even

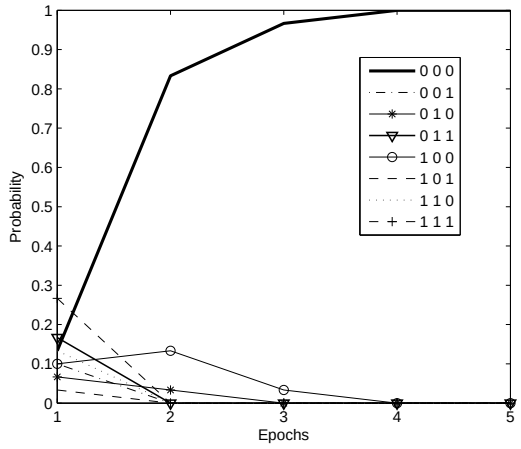

Fig. 2. A comparison of the probability a transfer function combination will yield the minimum error at the given epoch, if the network is trained only on combination (000) for the 3-bit parity problem.

while training has the potential to yield a lower performance measure. Furthermore, a learning algorithm which selects the best network at each epoch, and continues training with it could potentially result in a lower training error and/or a possible increase in convergence rate.

Next, we compute the difference $\Delta^{err} = Er_{\mathcal{N}} - \min(Er_{\Gamma(\mathcal{N})})$, and plot the results in Figure 4. The 3-bit parity problem shows a substantial difference between $\mathcal{N}$ and its opposite networks and the 6-bit version of the problem shows a smaller, yet increasing difference. However, the

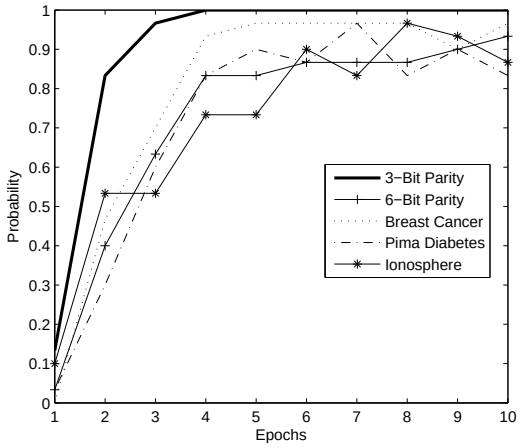

Fig. 3. Comparing the probability of transfer function combination (00...0) of yielding the lowest error for the five benchmark problems.

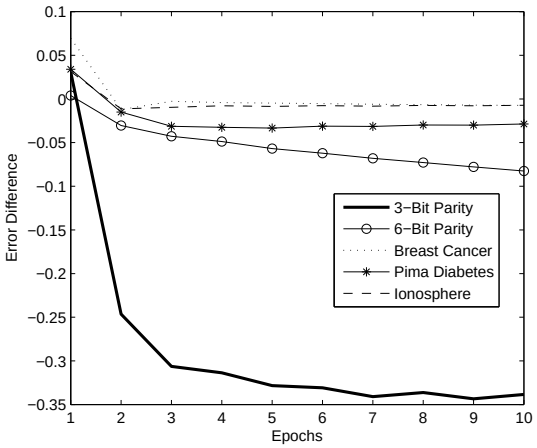

Fig. 4. Comparing the difference in error between the trained network and the opposite network with minimum error.

remaining three problems show very little difference between the network being trained and its opposites. For these latter three problems an opposite network is likely of about equal quality to the trained network, and it may be beneficial to consider switching the network being trained to the relevant opposite.

To examine the effect of training on the difference in rank of **J**, we calculate

$$\Delta^{rank} = \frac{\text{rank}(\mathbf{J}(\mathcal{N})) - \min(\text{rank}(\mathbf{J}(\Gamma(\mathcal{N}))))}{\min(|\mathbf{Z}|, |\mathbf{X}|)} \qquad (14)$$

where $\min(|\mathbf{Z}|, |\mathbf{X}|)$ represents the maximum[3] possible rank of the Jacobian matrix for each problem, respectively. The results are presented in Figure 5. The 6-bit parity and the Wisconsin breast cancer problems show a more rapid increase in the difference between the trained network, and the best opposite network with respect to rank. By the 10th epoch the other three problems show only a 2.0% difference in rank. So, for these latter three problems considering an opposite network (using only a rank criterion) is more likely to show an improvement than the former two problems.

[3]See Table I for the respective values.

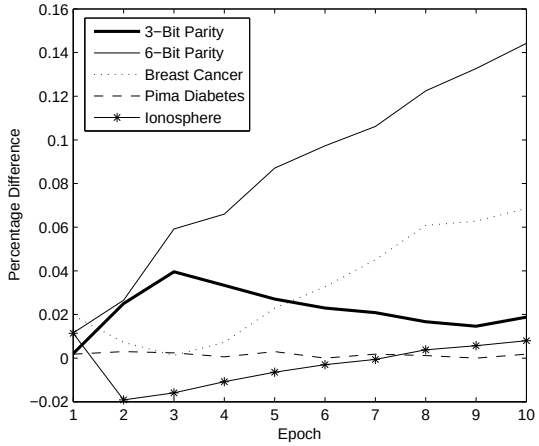

Fig. 5. Comparing the difference in rank between the trained network and opposite networks.

The final experiment will compare the difference in the mean condition κ of $\mathbf{H}$ over the 30 trials. To determine this we compute,

$$\Delta^{\kappa} = \kappa(\mathcal{N}) - \min(\kappa(\Gamma(\mathcal{N}))), \qquad (15)$$

where figure 6 plots these results. The Pima Diabetes and Ionosphere problems show a similar behavior, but the curve for the Pima data has most of its values > 0 which means that the trained network is not the best conditioned network. Since the Ionosphere results are mainly < 0 it is better conditioned that its opposite networks. The 6-bit parity and Wisconsin breast cancer data are both relatively close to having no difference between their trained and opposite networks, respectively. The 3-bit parity problem shows nearly no change in Δ^{κ}. These results show that even during training there are cases when it may be desirable to consider training an opposite network, especially if the training error shows little improvement.

V. Conclusions and Future Work

structures In this paper we examined the problem of ill-conditioning of neural networks and the potential impact of opposite transfer functions. We proved that OTFs are symmetrical transformations in weight space which yield unique input-output mappings under the assumption of a minimal random network as the base case for the transformation. Moreover, we were able to show that each of these networks has an equal probability of yielding the minimum error for a given problem before learning begins and without any prior information. We also described the potential changes OTFs can have on the rank of the Jacobian matrix as well as the conditioning of the Hessian.

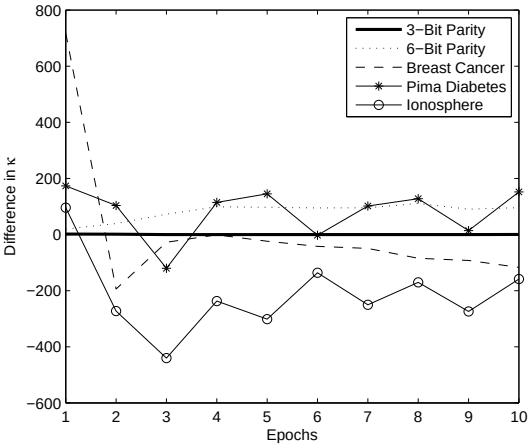

Fig. 6. A comparison of the difference in condition between the trained network and its opposite networks.

Our experiments confirmed the theoretical findings for pre-trained networks and also provided evidence for the consideration of OTFs during the early stages of training. Specifically, we experimentally showed that there is a probability that an opposite network may actually yield a lower error than a trained network, and that this probability is large enough to warrant consideration of opposite networks during training. The results for the rank of the Jacobian and condition of the Hessian also support the notion that opposite networks have desirable properties which are known to impact the accuracy and convergence rate of learning algorithms.

Future work will involve further theoretical and practical considerations. A deeper understanding of the learning trajectories of opposite networks, influence of weight initialization, network size and type of transfer function are important directions. Also, more experiments concerning different learning algorithms and problems is important. From this understanding, new strategies for utilizing OTFs or possibly new learning algorithms can be developed that lead to more accurate networks which are able to learn at a higher rate.

REFERENCES

[1] C. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2007.

[2] M. Riedmiller and H. Braun, "A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm," in *IEEE Conference on Neural Networks*, pp. 586–591, 1993.

[3] S. E. Fahlman, "Faster-Learning Variations on Backpropagation: An Empirical Study." Morgan Kaufmann, 1988.

[4] S. Haykin, *Neural Networks: A Comprehensive Foundation (2nd Edition)*. Prentice Hall, 1998.

[5] S. Saarinen, R. Bramley, and G. Cybenko, "Ill-Conditioning in Neural Network Training Problems," *SIAM Journal on Scientific Computing*, vol. 14, no. 3, pp. 693–714, 1993.

[6] N. Schraudolph, "Centering Neural Network Gradient Factors," in *Neural Networks: Tricks of the Trade* (G. Orr and K. R. Muller, eds.), pp. 207–226, Springer-Verlag, 1998.

[7] D. Nguyen and B. Widrow, "Improving the Learning Speed of 2-Layer Neural Networks by Choosing Initial Values of the Adaptive Weights," *IEEE Proceedings of the International Joint Conference on Neural Netowrks*, vol. 3, pp. 21–26, 1990.

[8] W. Duch and J. N., "New Neural Transfer Functions," *Applied Mathematics and Computer Science*, vol. 7, pp. 639–658, 1997.

[9] W. Duch and N. Jankowski, "Transfer Functions: Hidden Possibilities for Better Neural Networks," in *9th European Symposium on Artificial Neural Networks*, pp. 81–94, 2001.

[10] W. Duch and N. Jankowski, "Optimal Transfer Function Neural Networks," in *9th European Symposium on Artificial Neural Networks*, pp. 101–106, 2001.

[11] A. Thome and M. Tenorio, "Dynamic Adaptation of the Error Surface for the Acceleration of the Training of Neural Networks," vol. 1, pp. 447–452, 1994.

[12] N. Lagaros and M. Papadrakakis, "Learning Improvement of Neural Networks used in Structural Optimization," *Advances in Engineering Software*, vol. 35, no. 1, pp. 9–25, 2004.

[13] P. Chandra and Y. Singh, "An Activation Function Adapting Training Algorithm for Sigmoidal Feedforward Networks," *Neurocomputing*, vol. 61, 2004.

[14] P. Chandra and Y. Singh, "A Case for the Self-Adaptation of Activation Functions in FFANNs," *Neurocomputing*, vol. 56, pp. 447–545, 2004.

[15] M. Ventresca and H. R. Tizhoosh, "Improving the Convergence of Backpropagation by Opposite Transfer Functions," in *IEEE International Joint Conference on Neural Networks*, pp. 9527–9534, 2006.

[16] M. Ventresca and H. R. Tizhoosh, "Opposite Transfer Functions and Backpropagation Through Time," in *IEEE Symposium on Foundations of Computational Intelligence*, pp. 570–577, 2007.

[17] H. R. Tizhoosh, "Opposition-based Learning: A New Scheme for Machine Intelligence," in *International Conference on Computational Intelligence for Modelling, Control and Automation*, pp. 695–701, 2005.

[18] M. Shokri, H. R. Tizhoosh, and M. Kamel, "Opposition-based Q(lambda) Algorithm," in *IEEE International Joint Conference on Neural Networks*, pp. 646–653, 2006.

[19] H. R. Tizhoosh, "Opposition-based Reinforcement Learning," *Journal of Advanced Computational Intelligence and Intelligent Informatics*, vol. 10, no. 4, pp. 578–585, 2006.

[20] S. Rahnamayn, H. R. Tizhoosh, and S. Salama, "A Novel Population Initialization Method for Accelerating Evolutionary Algorithms," *(to appear) Computers and Mathematics with Applications*, 2006.

[21] S. Rahnamayn, H. R. Tizhoosh, and S. Salama, "Opposition-based Differential Evolution Algorithms," in *IEEE Congress on Evolutionary Computation*, pp. 7363–7370, 2006.

[22] S. Rahnamayn, H. R. Tizhoosh, and S. Salama, "Opposition-based Differential Evolution Algorithms for Optimization of Noisy Problems," in *IEEE Congress on Evolutionary Computation*, pp. 6756–6763, 2006.

[23] M. Ventresca and H. R. Tizhoosh, "Simulated Annealing with Opposite Neighbors," in *IEEE Symposium on Foundations of Computational Intelligence*, pp. 186–192, 2007.

[24] E. I. Barakova and L. Spaanenburg, "Symmetry: Between Indecision and Equality of Choice," *Biological and Artificial Computation: From Neuroscience to Technology*, 1997.

[25] A. Chen and R. Hecht-Nielsen, "On the Geometry of Feedforward Neural Network Weight Spaces," in *Second International Conference on Artificial Neural Networks*, pp. 1–4, 1991.

[26] A. M. Chen, H. Lu, , and R. HechtNielsen, "On the Geometry of Feedforward Neural Networks Error Surfaces," *Neural Computation*, vol. 5, no. 6, pp. 910–927, 1993.

[27] F. Albertini and E. Sontag, "For Neural Networks, Function Determines Form," *Neural Networks*, vol. 6, pp. 975–990, 1993.

[28] H. J. Sussmann, "Uniqueness of the Weights for Minimal Feedforward Nets with a Given Input-Output Map," *Neural Networks*, vol. 5, no. 4, pp. 589–593, 1992.

[29] F. Jordan and G. Clement, "Using the Symmetries of Multilayered Network to Reduce the Weight Space," in *IEEE Second International Joint Conference on Neural Networks*, pp. 391–396, 1991.

[30] S. Amari, H. Park, and T. Ozeki, "Singularities Affect Dynamics of Learning in Neuromanifolds," *Neural Computation*, vol. 18, pp. 1007–1065, 2006.

[31] P. van der Smagt and G. Hirzinger, "Solving the Ill-Conditioning in Neural Network Learning," in *Neural Networks: Tricks of the Trade* (G. Orr and K. R. Muller, eds.), pp. 193–206, Springer-Verlag, 1998.

[32] Y. Le Cun, J. S. Denker, and S. A. Solla, "Optimal Brain Damage," in *Advances in Neural Information Processing Systems*, pp. 598–605, 1990.

[33] D. N. A. Asuncion, "UCI machine learning repository," 2007.