Neural Networks Who Cheat And The Academics Who Can't Stop Loving Them

Shane Celis
CandNo. 79479

December 8, 2010

Abstract

This paper proposes a function ω that given a feedforward artificial neural network and an input will produce a metric for which inputs were most important to producing the network's output. The use of such a metric might aid casual detection of "cheating" neural networks that discriminate based on superficial input features. In addition, this paper exercises the ω function on a non-trivial problem to test its worth and the results are mixed.

1 Introduction

The mathematical topic I would like to tackle is analysing artificial feedforward neural networks that have been trained by supervised learning. Imagine a fixed neural network deployed in industry. It is difficult to diagnosis whether it is working properly on novel input , so how might one casually determine whether the network is working? This paper suggests one such approach.

From an engineering perspective, artificial neural networks offer great utility: One can approximate a function that is either poorly understood or may contain noise. For instance, one can train a network to perform text to speech as demonstrated by Sejnowski and Rosenburg with NETtalk[4]. Although one can easily compute a network's output for any given input, it is difficult to determine why the network produces a particular output in a larger sense. The network's workings at the finest level of detail are well understood, but the bigger picture of what features of the input a network is using to produce its output are opaque to casual inspection. This paper aims to propose and evaluate a technique to aid casual inspection.

This paper proposes a function ω that given a neural network and an input produces a metric for which inputs were most important to producing the network's output. The ω function would provide a valuable means of analysis. To test whether the ω function

is valuable, this paper exercises the ω function on a character recognition problem. We will describe the ω function and how it was constructed shortly. Let us begin with the problem we will use.

2 Method

The problem used to exercise the ω function is character recognition on a 16x16 grid with a small set of fonts: Helvetica, Courier, and Fixed. Letters 'a' thru 'z', both upper case and lower case characters are used. The network has $16 * 16 = 256$ inputs, k hidden, and 26 outputs. The activation function is a symmetric sigmoidal function. The monochrome images are transformed into a vector, each black pixel will be represented by a 1, each white pixel by a -1. This problem was inspired by Hofstadter [1] where he considers what makes an 'A' an A.

2.1 Experiment

The focus of this paper is not to build a good character recognition system, it is to exercise the proposed function ω on a problem to determine whether it is useful. To that end, a duplicate set of character images will be marked. The first pixel of the image 'a' is marked black, for image 'b' the second pixel is marked black, and so on. When a network is trained using the marked set, the network can recognize the characters via two means: 1) Look at all 256 inputs and determine which letter it is based on that—the hard way. 2) Look at the first 26 inputs and use the marks that always works irrespective of the font of the character—the easy way, or cheating. (One can think of the marked characters as a marked deck of playing cards that allow for cheating.) A sample of the input is shown in Figure 1.

Visual inspection of the ω function's output should allow one to tell whether a network cheated for that specific input. To be crystal clear about what constitutes success or failure for the ω function, assume we have or build a cheating network, a network that only looks at the first 26 inputs for instance. We want to see ω produce something similar to what is titled as "success" in Figure 2. The darker the more influential in the network's output, the lighter the less influential. The success image shows that the network essentially ignored most of the input and nearly based everything on the first pixel. In the failure case, the ω function does not provide a good metric for how the first input is being used to cheat. There are many other images that would constitute failure. Other failing images for this scenario would be an all black or all white image for instance.

The training set contains Helvetica and Courier fonts (total of $2 * 26 + 2 * 26 = 104$ patterns). The testing set contains the Fixed font (total of $2 * 26 = 52$ patterns). There is a unmarked training and testing set. And there is a marked training and testing set. The marked testing set also contains 26 images that are blank except for the mark for 'a' thru 'z'.

The training set will be run for different values of hidden neurons k. Then the proposed ω function will be used on cases where

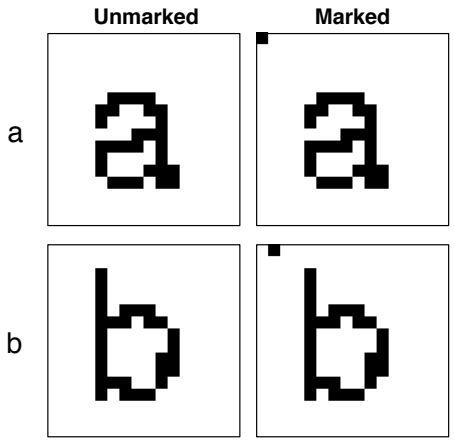

Figure 1: Sample of letters in Helvetica font

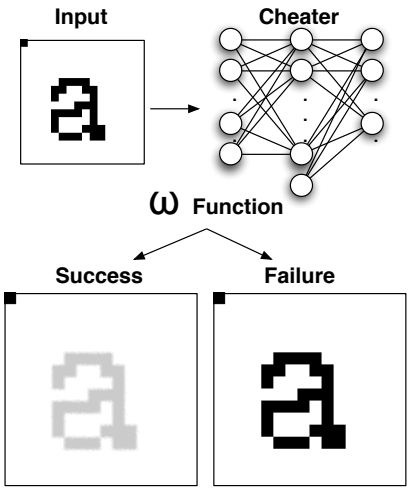

Figure 2: Criterion for judging ω function's utility.

the network can only recognized the marked letter but not the unmarked one. On the basis of those visual images, we can determine whether ω is useful.

2.2 Implementation (Informal)

The network was trained using backpropagation. I began to implement backpropagation in Mathematica, which I have included as 'neural-network.nb', but for the sake of time I used the Fast Artificial Neural Network (FANN) Library [2] with Python bindings. The python code is included.

2.3 Notation

Before elaborating on how the ω function is constructed, let us define the notation to represent neural networks. The notation here will use a matrix formulation similar to Rojas'[3, pp 165].

$$\begin{array}{ll} \mathbf{s^i} \in \mathbb{R}^n & \text{input vector} \\ \mathbf{W_1} \in \mathbb{R}^{n \times k} & \text{connection matrix} \\ \mathbf{W_2} \in \mathbb{R}^{k \times m} & \text{connection matrix} \\ \mathbf{s^h} \in \mathbb{R}^k & \text{hidden layer output} \\ \mathbf{s^o} \in \mathbb{R}^m & \text{output layer output} \\ \mathbf{f_a} & \text{activation function} \end{array} \tag{1}$$

In order to take care of neuron thresholds, we use biases and an extension operator for vectors. The vector $\mathbf{s^i} = (s_1^i, \ldots, s_n^i)$ extended is $\hat{\mathbf{s}}^\mathbf{i} = (s_1^i, \ldots, s_n^i, 1)$. The connection matrices are also extended to hold an extra row of elements—the weights for the bias. The superscript in $\mathbf{s^i}$, $\mathbf{s^h}$, and $\mathbf{s^o}$ is purely for

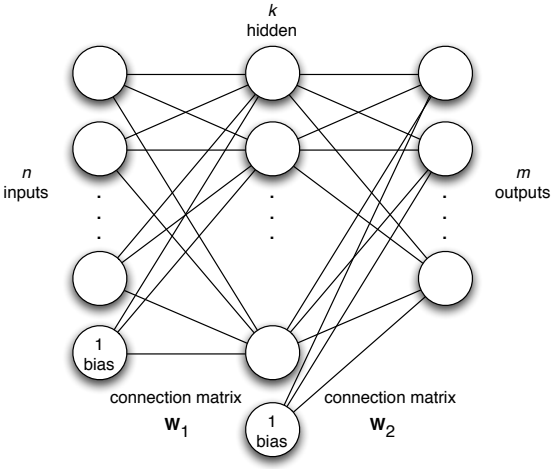

Figure 3: Three-layer network

notation purposes to discriminate between input, hidden, and output respectively.

$$\begin{array}{ll} \hat{\mathbf{s}}^{\mathbf{i}} \in \mathbb{R}^{n+1} & \text{extended input vector} \\ \overline{\mathbf{W}}_{\mathbf{1}} \in \mathbb{R}^{(n+1)\times k} & \text{ext. matrix} \\ \overline{\mathbf{W}}_{\mathbf{2}} \in \mathbb{R}^{(k+1)\times m} & \text{ext. matrix} \end{array} \tag{2}$$

To exercise the network we have the following equations:

$$\begin{aligned} \mathbf{s}^{\mathbf{h}} &= \mathbf{f}_{\mathbf{a}}(\overline{\mathbf{W}}_{\mathbf{1}}{}^{\mathbf{T}}\hat{\mathbf{s}}^{\mathbf{i}}) \\ \mathbf{s}^{\mathbf{o}} &= \mathbf{f}_{\mathbf{a}}(\overline{\mathbf{W}}_{\mathbf{2}}{}^{\mathbf{T}}\hat{\mathbf{s}}^{\mathbf{h}}) \end{aligned} \tag{3}$$

Now that the notation is clear, we can address the construction of ω.

2.4 Constructing the Omega Function

We want a function ω that given an input $\mathbf{s}^{\mathbf{i}}$ and access to the weights $\overline{\mathbf{W}}_{\mathbf{1}}$ and $\overline{\mathbf{W}}_{\mathbf{2}}$, will produce a vector in $\mathbb{R}^n$.

$$\omega : \mathbb{R}^n \rightarrow \mathbb{R}^n \tag{4}$$

This paper will build the proposed ω function from the ground up, as that will be the easiest way to provide justification. The activation function is assumed to be monotonically increasing and $\mathbf{f}_{\mathbf{a}}(\mathbf{0}) = \mathbf{0}$, which is true of the symmetric sigmoidal function used in this paper.

2.4.1 Perceptron-ish

Consider a two-layer network, with one output neuron as shown in Figure 4. The network is very similar to a perceptron.

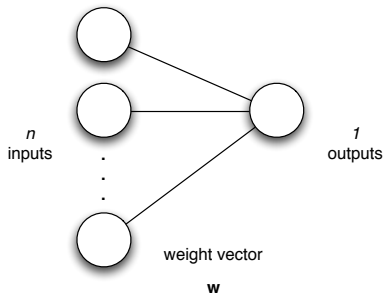

Figure 4: Two-layer network with one output

What is the ω for this perceptron-ish network? Let us call it ω_a for this specific network. There are many different functions that could work. A natural choice is to use the Hadamard product on the weight vector $\mathbf{w}$ and the input vector $\mathbf{s}^{\mathbf{i}}$. The Hadamard product (5) is an element-wise multiplication of two matrices or vectors of the same size.

$$\mathbf{H}(\mathbf{A}, \mathbf{B})_{\mathbf{i,j}} = \mathbf{A}_{\mathbf{i,j}} \cdot \mathbf{B}_{\mathbf{i,j}} \tag{5}$$

One can think of the Hadamard product for vectors as an inner-product without the summation step. It is a natural choice because the neural network is computing $\mathbf{w} \cdot \mathbf{s^i}$ to find the net excitation.

One property we can expect from ω_a is that if the weight vector is $\mathbf{0}$, then $\omega_a(\mathbf{s^i})$ ought to be $\mathbf{0}$ too. Justification being that if the input has no bearing on the network's output, every input has the same impact—zero. And the Hadamard product satisfies this property.

$$\omega_a(\mathbf{s}, \mathbf{w}) = \mathbf{H}(\mathbf{s}, \mathbf{w}) \tag{6}$$

2.4.2 Two Outputs

Consider a two-layer network with 2 outputs as shown in Figure 5.

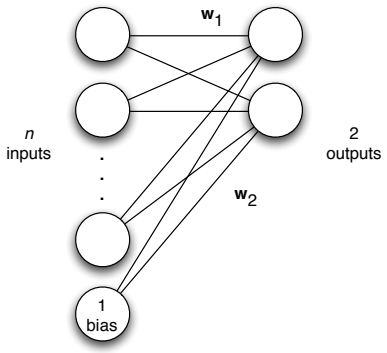

Figure 5: Two-layer network with 2 outputs

What is the ω for this network? Let us call it ω_b for this specific network. We can use ω_a to produce a n-length vector for both output neurons. The problem is that we have two of them now. We need to aggregate them somehow.

$$\omega_b(\mathbf{s}) = f(\omega_a(\mathbf{s}, \mathbf{w_1}), \omega_a(\mathbf{s}, \mathbf{w_2}))$$

Our task is to choose an f. What properties of ω_b can we rely on to guide our choice? Consider the case where $\mathbf{w_2}$ equals $\mathbf{0}$.

$$\omega_a(\mathbf{s}, \mathbf{w_2}) = \omega_a(\mathbf{s}, \mathbf{0}) = \mathbf{0}$$
$$\omega_b(\mathbf{s}) = f(\omega_a(\mathbf{s}, \mathbf{w_1}), \mathbf{0})$$

If $\mathbf{w_2} = \mathbf{0}$, it should degenerate into our first case.

$$\omega_b(\mathbf{s}) = \omega_a(\mathbf{s}, \mathbf{w_1})$$

That $f(\omega_a(\mathbf{s}, \mathbf{w_1}), \mathbf{0}) = \omega_a(\mathbf{s}, \mathbf{w_1})$ suggests f should be preserve the additive identity for zero, which a multiplicative operation like the Hadamard multiplication does not do. Let us choose f to just be a simple summation, which extrapolates to networks with m outputs.

$$\omega_b(\mathbf{s}, \mathbf{w_1}, \ldots, \mathbf{w_m}) = \sum_{i=1}^{m} \omega_a(\mathbf{s}, \mathbf{w_i}) \tag{7}$$

2.4.3 Three Layers

Let us now consider the case where we have a three layer network with n inputs, k hidden, and m output neurons as shown in Figure 3. What is the ω function for this network? It is our last ω so we need not rename it. We can use ω_b to produce a reasonable result between the input and the hidden layer.

$$\mathbf{x_1} = \omega_b(\mathbf{s^i}, \mathbf{W_1})$$

Similarly, for the hidden and the output layer,

$$\mathbf{x_2} = \omega_b(\mathbf{s^h}, \mathbf{W_2})$$

but $\mathbf{x_1}$ and $\mathbf{x_2}$ are different sizes. We have a composition problem again, and need find some function g.

$$\omega(\mathbf{s^i}) = g(\omega_b(\mathbf{s^i}, \mathbf{W_1}), \omega_b(\mathbf{s^h}, \mathbf{W_2}))$$
$$\omega(\mathbf{s^i}) = g(\mathbf{x_1}, \mathbf{x_2})$$

We can find a constraint for g by considering the case where $\mathbf{W_2} = \mathbf{0}$. In this case, despite having $\mathbf{x_1} \neq \mathbf{0}$, the effects of the input never make it past the hidden layer, so the ω of the input should be zero.

$$\omega(\mathbf{s^i}) = \mathbf{g}(\mathbf{x_1}, \mathbf{x_2}) = \mathbf{g}(\mathbf{x_1}, \mathbf{0}) = \mathbf{0}$$

This suggests g is a multiplicative function, so perhaps the hadamard product will work. However, $\mathbf{x_1}$ and $\mathbf{x_2}$ are not necessarily the same dimensions. The matrix $\mathbf{W_1}$ can bridge that gap and account for the weights in the network.

$$g(\mathbf{x_1}, \mathbf{x_2}) = \mathbf{H}(\mathbf{x_1}, \mathbf{W_1}\ \mathbf{x_2})$$

Finally, we have an expression for ω (8) that works for a three-layer network, and can be extended to work on an n-layer network. Note that it never computes anything with the output $\mathbf{s^o}$ or uses the activation function $\mathbf{f_a}$. However, the activation function must be monotonically increasing and $\mathbf{f_a}(\mathbf{0}) = \mathbf{0}$ otherwise this construction of the omega function is invalid.

$$\omega(\mathbf{s^i}) = \mathbf{H}(\mathbf{x_1}, \mathbf{W_1}\ \mathbf{x_2}) \tag{8}$$

This paper does not mean to suggest that the proposed ω function is the only one of its kind. There are many different kinds of functions that would respect the properties outlined.

3 Results

Figures 6 and 7 show the accuracy of the networks for both the marked and unmarked training and testing sets. Figure 8 demonstrates the behavior that we are interested in detecting, cheating. From the graph we can see that the network trained with the marked set cannot recognize the unmarked training set even though they only differ by just one input. Let us examine one of the unmarked characters that it does not recognize, the character 'y' and 'f', which will show mixed results for the proposed function's utility.

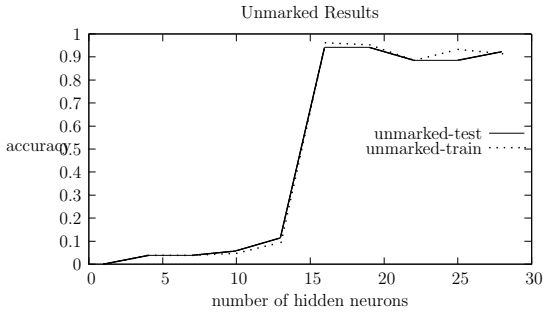

Figure 6: Accuracy for the unmarked training and testing sets, which shows that approximately 16 hidden neurons are required to perform the task.

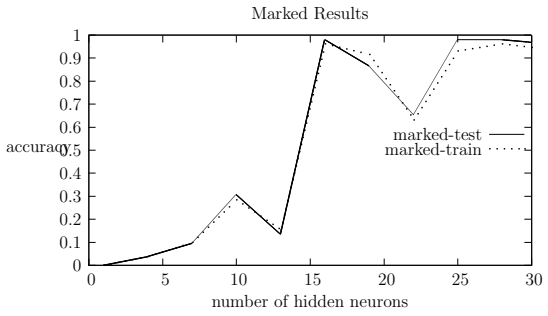

Figure 7: Accuracy for the marked training and testing sets shows very similar performance to the unmarked set in Figure 6

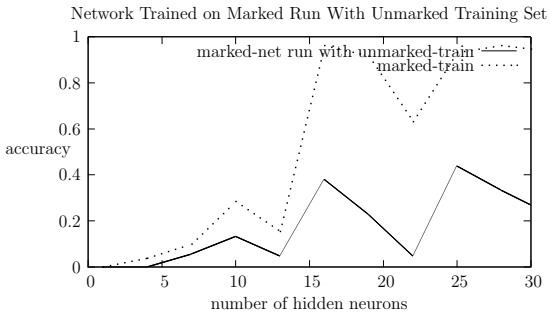

Figure 8: Performance of the network trained on marked images with the unmarked images shows that the network can only recognise marked images with high accuracy, so the marks are significant. This could be due to cheating or overfitting.

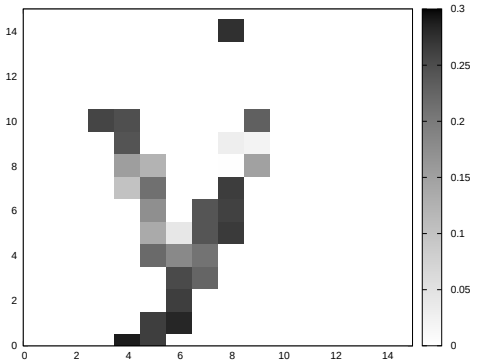

Figure 9: Result of ω with cheater network that can only recognise the marked 'y'. Unfortunately, the image does not suggest that ω would help ascertain whether this network was a cheating.

Figure 9 shows the output of ω for a network with 16 hidden neurons trained with the marked set of characters. It recognizes

Helvetica 'y' with the mark, but not without it. This is not the result that was hoped for. However, Figure 10 for character 'f' compares the output of the ω function for a cheating and non-cheating networking, and demonstrates the kind of image hoped for. So the results are mixed.

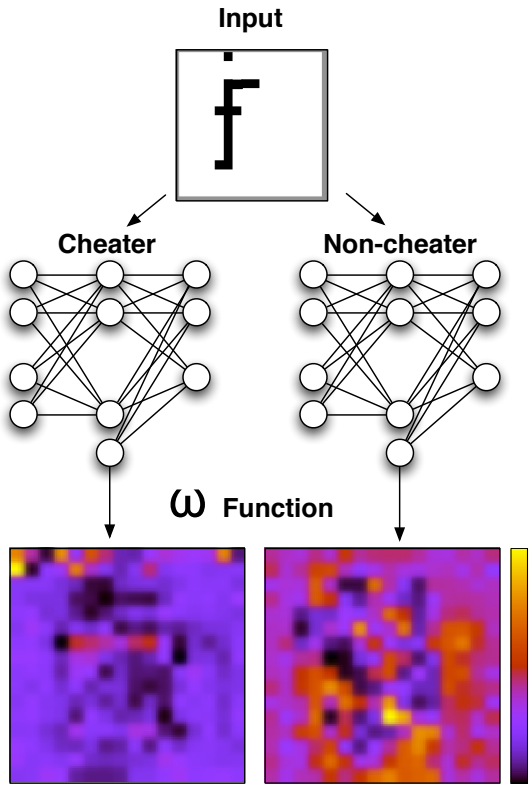

Figure 10: Shows a marked input that both networks recognise as 'f'. However, the output of the ω function displays what the network "looked" at to make its decision. The left image shows intensity on the top two rows where the marks are located, indicating it is a cheater. The right image shows intensity where the pixels of the characters typically are, indicating it is not cheating.

4 Discussion

Figure 9 does not support the assertion that the ω function would be helpful in detecting

cheating networks. It was hoped that the 'y' would be very washed out, but as it is the dot does not *appear* to strongly influence the network's output. In fact, there is a darker spot on the tail of the 'y'. However, Figure 10 does support the assertion, so it appears that some parts of the idea may have merit. It would be interesting to find out what makes the two cases different; it may be due to network sensitivity.

Inspecting further, the networks trained with unmarked data was expected to be insensitive to the dot; however, this was not the case. Using the network with 16 hidden neurons trained with unmarked data, it could not detect the marked 'y'. The sensitivity of these networks may be clouding the analysis conducted here.

5 Future Work

Rather than trying to train a cheater, it might have been worthwhile to create a cheating network by hand. One would not be susceptible to network sensitivites in that case. However, this may only trivially support the utility of the ω function. Another kind of analysis that might be interesting is not how much "weight" a given input has on the network's output as ω tries to determine, but how insensitive the network is to perturbations in its input. One can imagine a color map that shows how readily the input can vary without changing the output of the network. That may provide the kind of insight sought but not necessarily found with the ω function.

References

[1] Douglas R. Hofstadter. On seeing A's and seeing As. *Stanford Hum. Rev.*, 4(2):109–121, 1995.

[2] Steffen Nissen. Neural Networks Made Simple, Software 2.0, 2005.

[3] Raúl Rojas. *Neural networks : a systematic introduction*. Springer-Verlag, Berlin, 1996.

[4] Terrence J. Sejnowski and Charles R. Rosenberg. NETtalk: a parallel network that learns to read aloud. pages 661–672, 1988.