Neuron Selectivity as a Biologically Plausible Alternative to Backpropagation

C.J. Norsigian
Department of Bioengineering
University of California, San Diego
cnorsigi@eng.ucsd.edu

Jesper Pedersen
Department of Physics
University of California, San Diego
jrpeders@ucsd.edu

Vishwajith Ramesh
Department of Bioengineering
University of California, San Diego
vramesh@eng.ucsd.edu

Hesham Mostafa
Institute for Neural Computation
University of California, San Diego
hmmostafa@ucsd.edu

Abstract

In this paper, we aim to develop alternative methods to backpropagation that more closely resemble biological computation. While backpropagation has been an extremely valuable tool in machine learning applications, there is no evidence that neurons can back propagate errors. We propose two methods intended to model the intrinsic selectivity of biological neurons to certain features. Both methods use selectivity matrices to calculate error and to update synaptic weights in different ways, either by comparing neuronal firing rate to a threshold - firing rate algorithm - or computing error from a generated score - scoring algorithm. We trained and tested networks that used either of the two algorithms with the MNIST database. We compared their performance with a multilayer perceptron network with 3 hidden layers that updated synaptic weights through typical error backpropagation. The backpropagation algorithm had a test error of 1.7%. Networks that updated synaptic weights based on the scoring method gave a test error of 2.0%, and networks using the firing rate method 4.3%. We were thus able to develop more biologically plausible models of neural networks in the brain while obtaining performance comparable to typical backpropagation algorithms.

1 Introduction

Error backpropagation used in conjunction with gradient descent is a common method to train artificial neural networks. The method consists of two steps. First, during forward propagation, the input is passed forward through the hidden layers of a multilayer neural network in order to finally obtain a predicted output. Specifically, the output of each layer is weighted and an activation function (such as the rectified linear unit activation function) applied, which is then passed as input into the next layer. The output of the last layer is the predicted label. In order to increase the accuracy of the neural network - that is, to increase the probability that the predicted output is what we would expect for a given training input - we backpropagate errors and update the weights between layers. In particular, error backpropagation involves:

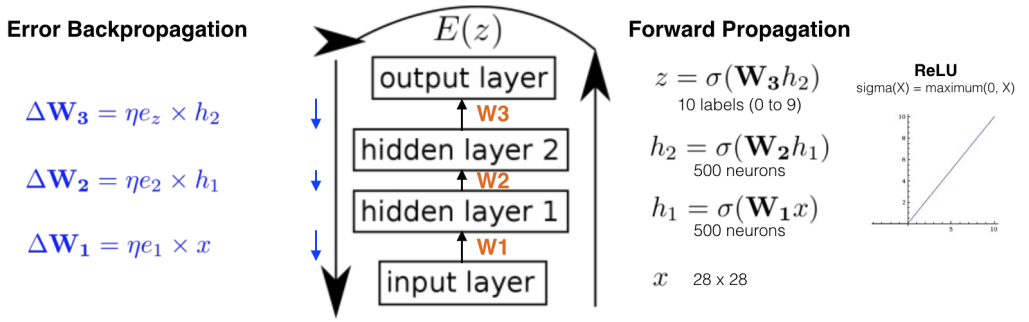

Figure 1: Flow of calculations involved in forward propagation and error backpropagation in a neural network with 2 hidden layers. The input, x, is a 28x28 pixel hand drawn digit. It is weighted by W_1 and passed into Hidden Layer 1. h_1 is calculated by applying the rectified linear unit (ReLU) activation function on the weighted input into Hidden Layer 1. The output of Hidden Layer 2 h_2 and of the Output Layer z are similarly calculated. A loss $E(z)$ is calculated at the Output Layer and is used to calculate error terms e_z, e_2, and e_1 that are used to update the weights between layers, in proportion to the learning rate η.

1. Using the predicted label y at the last layer of the network and expected training label $\hat{y}$ to calculate an error E or loss. Typically, the mean squared error equation is used:

$$E = \frac{1}{2} * (y - \hat{y})^2$$

2. Calculating the gradient of this loss $E(z)$ with respect to the output z of the neural network and propagating the resultant error term e_z backwards to calculate similar error terms for the previous layers. For a neural network with 4 total layers (2 hidden, 1 input, and 1 output) and 4 weights W (**Figure 1**):

$$e_z = \sigma'(W_3 h_2) \circ dE/dz$$
$$e_2 = \sigma'(W_2 h_1) \circ W_3^T e_z$$
$$e_1 = \sigma'(W_1 x) \circ W_2^T e_2$$

3. Updating each weight W of the neural network by ΔW by taking steps proportional to the negative of the gradient to minimize error E. The step size η is the learning rate. For a neural network with 4 layers (**Figure 1**):

$$\Delta W_3 = \eta e_z \times h_2$$
$$\Delta W_2 = \eta e_2 \times h_1$$
$$\Delta W_1 = \eta e_1 \times x$$

4. Repeating the above steps over several passes to update weights to minimize error and maximize prediction accuracy of the neural net (1; 5).

There is no biological evidence that neurons compute error to improve performance in the way described above. In particular, error backpropagation involves a weight symmetry with forward propagation. During forward propagation, the output of one layer is weighted by a weight matrix W and passed as input into the next layer. During backpropagation, the error is passed backwards and weighted by the transpose of the weight matrix W^T, as can be see in Step 2 above. This exact symmetry between forward propagating signal and backwards propagating error is unlikely to be found in nature; there is no evidence in the brain that the connection strength from Neuron X to Neuron Y is the same as the connection strength from Neuron Y to Neuron X.

There is evidence in the literature that individual neurons are selective to a number of categories in a task dependent manner. Research in the IT area of monkeys has demonstrated that when presented

with a categorization task, neurons can be predisposed to various categories involved in performing the task. That is, certain neurons have greater activity for categories deemed important for the given task, such as the eyes of a face if the task involves some decision about the eyes (2; 3; 4).

We seek to use this property of neuron selectivity to develop alternatives to the standard technique of backpropagation. For each neuron in the neural network, we model its selectivity to each possible category (output label) as either +1 if the neuron is selective, -1 if it is anti-selective, or 0 if it is agnostic. Each layer in a multi-layer network will generate its own unique error signal based on the feature selectivity of neurons in that layer without propagating e_z backwards from the output layer to the input layer. The use of self-generating error signals with unique selectivity masks could provide a more biologically realistic alternative to the commonly used back propagation structure.

2 Methods

Each neuron in a layer of the network has a selectivity profile, a randomly generated vector of +1, 0, or -1 for each possible output category. We developed two algorithms - firing rate and scoring - that both use this selectivity profile to generate error signals but do so in slightly different ways. Note that forward propagation and the weight update is the same for backpropagation, firing rate, and scoring; the difference between the three methods is in how the error used to update the weights is calculated.

Neural networks implementing backpropagation, firing rate, or scoring to update weights were written in Python. The code is provided as Supplementary Information.

2.1 Firing Rate Algorithm

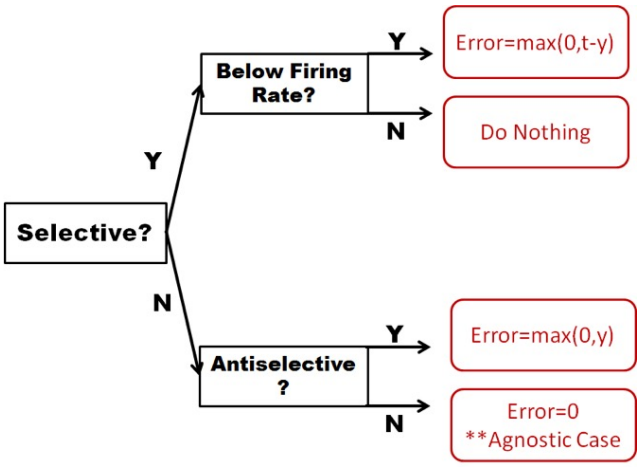

Figure 2: Decision tree for the error attributed to each neuron for the target firing rate method. Each neuron generates its own error based whether it is selective, and if that is the case, then whether it fires above or below the set threshold, t. If it is anti-selective, then an error is generated. Finally, if the neuron is agnostic, it has an associated error of 0. The neuron's firing activity is y.

In the firing rate method, a global "firing rate parameter" is defined as threshold for neural activity. For a given layer, when a selective neuron in that layer fires below this threshold, the weight applied to the input into that neuron (during forward propagation) is updated to push the selective neuron's activity past the threshold. If a selective neuron fires above the set threshold, the weight is not updated as the neuron is already performing adequately. For anti-selective neurons, the weights are updated so that the activities of the neurons are pushed to zero. For agnostic neurons, no updates occur as these neurons are indifferent to the current input. This flow of decisions for each neuron is depicted in **Figure 2**.

2.2 Scoring Algorithm

In the scoring method, the activity of each hidden layer is multiplied with the randomly generated selectivity matrix for all the 500 neurons in that layer, M. Thus, the score is defined as $M * y$. This generates a score vector for that layer. Note that for the scoring method, there is no output layer; each hidden layer is capable of producing an output, a predicted label. The predicted label for each layer is then compared to the expected label and a cross entropy error is generated. This error is used to only update the weights of the inputs into that layer, as shown in **Figure 3**, in order to ultimately

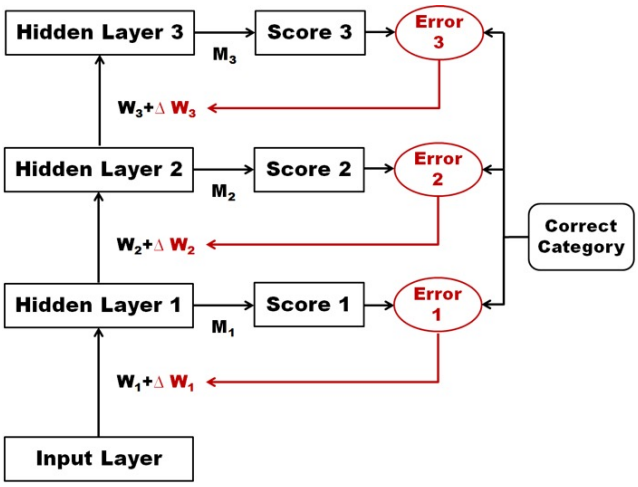

Figure 3: Flowchart for the scoring algorithm. Each layer gives a predicted label. Then each layer calculates its own error, by determining the cross entropy loss of that layer's prediction (from the score matrix corresponding to that layer) and the expected label.

minimize the error at each layer similar to what is done during Step 3 in the description of error backpropagation.

2.3 Evaluation of Algorithms

We used the MNIST database to train and test our algorithms. The MNIST database contains a series of 28-by-28 pixel images of handwritten digits between 0 and 9 (10 output categories). We used the 60,000 training examples exclusively for training, and tested on the 10,000 test examples (6).

Using the MNIST results as a benchmark, we evaluated the 3 different approaches to generating errors. We used neural nets with 3 hidden layers, each with 500 neurons. We used gradient descent, with mini-batches of 50 examples (5). We evaluated the training and test errors, calculated by adding the number of correct predictions (of occurrences where the predicted and expected labels were the same) and dividing by the total number of examples used to train or test.

Each neural net was optimized for 50 epochs where one epoch is equal to a full run through of our training set (that is $60,000/50$ mini-batches). After each epoch, the learning rate was halved to avoid stepping past the local minimum in the gradient of the error function.

3 Results

The results of standard backpropagation and the firing rate method are shown in **Figure 4**. After 10 epochs the training error for standard backpropagation is approximately 0% and the test error saturates at 1.7%. For a defined firing rate parameter of 0.8, the training error saturates at 3.5% and test error at 4.3% after roughly 50 epochs.

For the scoring algorithm, the results are shown in **Figure 5**. Here, each layer itself generates an output based on its scoring mask. The first layer saturates first after about 14 epochs with a training error of 0% and a test error of 2.0%. The second layer saturates to the training and test errors of the first layer after 17 epochs. Finally, the third layer saturates after 26 epochs to a training error of 0.0% and a test error of 2.0%.

4 Discussion

We note the effectiveness of the backpropagation algorithm. The training and test errors both fall to their steady-state values quickly, after roughly 10 epochs; the last 40 epochs do not change the errors significantly (**Figure 4**). We also note that the firing rate algorithm does not perform as effectively as backpropagation. In fact even after 50 epochs, it appears that the error curves have not yet reached their steady state value. The test error value that the firing rate algorithm gives after 50 epochs is 4.3%, 2.6% greater than the 1.7% test error for backpropagation. While the firing rate algorithm does perform well, it does not perform as well as backpropagation.

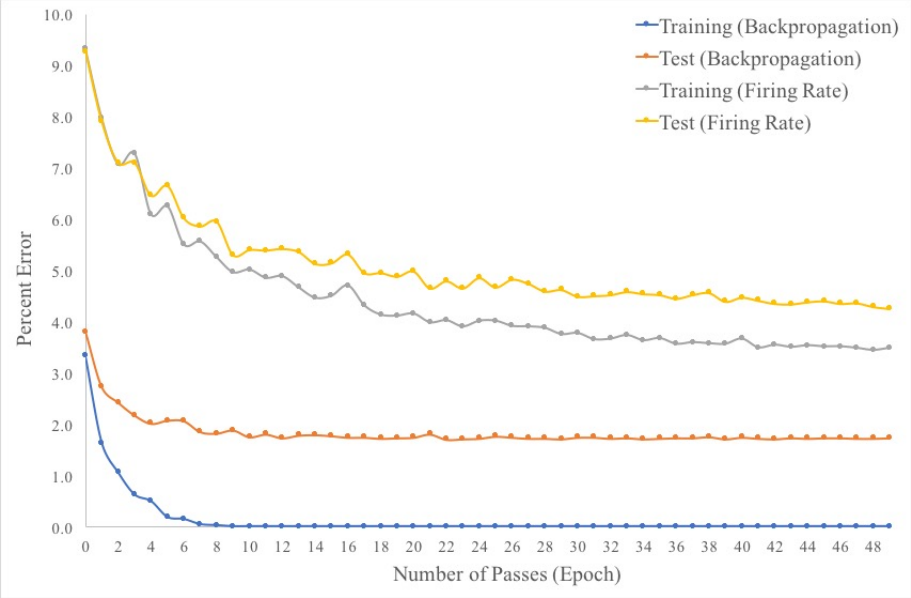

Figure 4: Training (blue) and test (orange) errors for back propagation and training (purple) and test (yellow) errors for the firing rate method, both for 50 epochs. The two methods are used in a neural network consisting of 3 hidden layers each with 500 neurons. The training error for backpropagation is approximately 0% after 10 epochs, at which point the test error saturates as well, to a value of 1.7%. For the firing rate method, we set a target threshold of 0.8. Weights and selectivity masks were initialized randomly. After 50 epochs, the training error saturates to a value of 3.5% and the test error to a value of 4.3%.

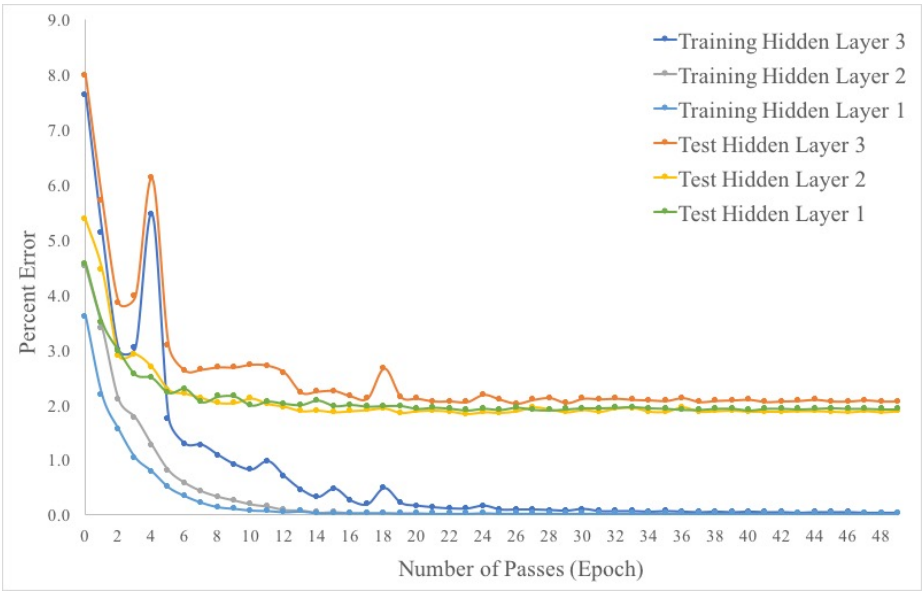

Figure 5: Training (blue, gray, light blue) and test errors (red, yellow, green) for the 3 hidden layers of the network shown in **Figure 3** over 50 epochs. The training and test errors associated with the output of the first layer converges after 10 epochs, with the errors associated with the other two layers following. Final training error is 0.0% with a test error of 2.0%.

On the other hand, the scoring method alternative produces a test error of 2.0% (at the top layer), just 0.3% greater than the test error for backpropagation (**Figure 5**). It performs better than the firing rate

algorithm and produces results comparable to backpropagation. The scoring method algorithm is thus a biological alternative to backpropagation that performs approximately as well.

We note that the first hidden layer for the neural network that uses the scoring method saturates to its steady-state error value first, after roughly 10 epochs for both training and test error curves. The second hidden layer saturates at roughly 12 epochs and the third hidden layer at roughly 26 epochs (**Figure 5**). The error for the top layer, Hidden Layer 3, cannot converge until the two layers beneath it stabilize, since the top layer is dependent on the bottom two layers for its input. In other words, only once the errors associated with Hidden Layer 1 and Hidden Layer 2 converge can the errors for Hidden Layer 3 converge. The same applies for Hidden Layer 2, which is dependent on the output of Hidden Layer 1 for its input. Essentially, Hidden Layer 2 has to "wait" for the output of Hidden Layer 1 to be optimized such that the error associated with Hidden Layer 1 is minimized. This optimized output is passed into Hidden Layer 2 as input (**Figure 5**). It is for this reason that the error curves for Hidden Layer 1 saturate first, then those for Hidden Layer 2, and lastly those for Hidden Layer 3. Because the scoring method provides the opportunity to examine the predicted label at each level of the network, it would be an interesting avenue for further research to see how performance varies by adjusting the number of layers.

During the first 20 epochs, the error curves for the scoring algorithm are not as smooth as the curves for backpropagation. That is, the descent of the former curves is uneven. This is indicative of a learning rate that has not been optimized for the scoring algorithm. For the scoring and backpropagation methods, we reduced the learning rate by half after each epoch. While this seems to have worked well for backpropagation (the backpropagation error curves are much smoother), halving the learning rate is not ideal for the scoring method. In fact, as can be seen from Figure 5, the training and test error curves for Hidden Layer 3 jump up at 6 and 18 epochs. Optimizing the learning rates may lead to better accuracies that reduce the 0.3% difference in error between scoring and backpropagation. Rather than halving the learning rate at the end of each epoch, we can consider using a step decay to reduce the learning rate.

Another area we intend to explore in the future is to test the robustness of the three error calculation methods. We can measure how performance suffers (how the training and test error curves look) in response to randomly removing a certain percentage of neurons (25%, 50%, 75%, etc.) at each hidden layer in the three networks. Such a study may reveal whether our biological alternatives are more robust than error backpropagation.

We can also optimize our selectivity masks in addition to the synaptic weights. Given the importance of the selectivity masks to both the firing rate and scoring methods, randomly generating the selectivity masks initially and updating them to minimize error (as we update the weights) may improve the accuracies of the firing rate and scoring algorithms.

Lastly, we intend to see whether we can define a selectivity matrix for backpropagation based on the weight of the input into each neuron in each layer. We propose that by analyzing the resulting weights after backpropagation, a selectivity matrix similar to those defined in our other methods can be generated based on the network's activity. We may thus be able to compare selectivity matrices for backpropagation with matrices for firing rate and scoring. It would be interesting to see whether backpropagation does in fact result in neuron selectivities that are comparable to those of the more biological firing rate and scoring algorithms. Alternatively, we could employ the standard methods for evaluating selectivity, which present examples from the different categories and then define a selectivity metric as a function of the neuron's responses to the examples from each category.

5 Conclusion

In this project, we implemented two new techniques for error calculation that are alternatives to error backpropagation using gradient descent: firing rate and socring. We evaluated networks utilizing each of the three methods for error calculation by training and testing on the standard MNIST dataset.By analyzing the training and testing error of these networks we show that alternative methods inspired by biology may be viable alternatives to backpropagation. Most interestingly, the scoring method came reasonably close to error backpropagation in performance (2.0% versus 1.7%). There are also opportunities to improve the method further in future work, by annealing the learning rate in different ways or by dynamically updating the selectivity matrices to minimize error.

6 Acknowledgements

We would like to thank Professor Cauwenberghs for running this valuable course. We would also like to thank Hesham Mostafa for his invaluable guidance and mentorship on this project.

References

[1] Balakrishnan, Divya, and Sadasivan Puthusserypady. "Multilayer perceptrons for the classification of brain computer interface data." Proceedings of the IEEE 31st Annual Northeast Bioengineering Conference, 2005. IEEE, 2005.

[2] Kiani, Roozbeh, et al. "Object category structure in response patterns of neuronal population in monkey inferior temporal cortex." Journal of Neurophysiology 97.6 (2007): 4296-4309.

[3] Sigala, Natasha, and Nikos K. Logothetis. "Visual categorization shapes feature selectivity in the primate temporal cortex." Nature 415.6869 (2002): 318-320.

[4] Sigala, Natasha, F. Gabbiani, and N. K. Logothetis. "Visual categorization and object representation in monkeys and humans." Journal of Cognitive Neuroscience 14.2 (2002): 187-198.

[5] Widrow, Bernard, and Michael A. Lehr. "30 years of adaptive neural networks: perceptron, madaline, and backpropagation." Proceedings of the IEEE 78.9 (1990): 1415-1442.

[6] Yann LeCun, Corinna Cortes and Christopher J.C. Burges. "The MNIST database of handwritten digits".
From: http://yann.lecun.com/exdb/mnist/.