Gesture Recognition using Neural Networks

Jeremy Smith
Department of Computer Science
George Mason University
Fairfax, VA
Email: jsmitq@masonlive.gmu.edu

ABSTRACT

A gesture recognition method for body gestures is presented. Image recognition is accomplished by using a color tracking algorithm. Results of the image recognition process are then sent through a feed-forward neural network and trained using a backpropagation algorithm. Training is done over an ensemble using different randomized initial weights to reduce local minima error. Results of testing the trained network are then presented and compared to other methods.

KEYWORDS

Gesture recognition, image recognition, neural network, ensemble learning

1. INTRODUCTION

Gesture recognition is a heavily studied area of computer science in which a human is able provide inputs to a computer system using hand gestures, body gestures, or some other form of visual signal. This subject has become increasingly popular in recent years as computer processing power has increased to a point where it is able to process visual images in real-time. This allows one to interact more naturally with computers than by using a mouse, keyboard, or other mechanical input device.

The methods behind gesture recognition currently live on the cutting edge of computer science. They often incorporate multiple techniques from image processing, computer vision, pattern recognition, and machine learning. These subjects have been studied extensively due to the wide range of possible real-world applications.

We have already seen instances in the consumer market where gesture recognition has been put to use. A sign language interpreter can read hand gestures from a user and output corresponding letters and words from a specified language. Video game consoles have embraced the use of body gestures as a form of interaction within a virtual game environment. Facial gesture recognition has also been used to recognize emotion.

In this experiment, the focus is on recognizing gestures made by the upper body of the subject. For simplicity, the gestures used will be taken from the official NFL hand signals: Holding, Timeout, Offside, and Touchdown (as shown in Figure 1). Images will be sent through the system and the appropriate gesture will be recognized and returned.

Figure 1: Four gestures used in this experiment

2. RELATED WORK

Much research has been done on the topic of gesture recognition, mainly in the area of hand gestures. Philip A. Harling at the University of York did extensive research on the subject with both static hand gestures and dynamic hand gestures, in both cases using feed-forward neural networks (with the dynamic

experiment using a much larger set of input nodes and hidden nodes) [1]

A gesture recognition method for recognizing Japanese sign language is presented by Murakami and Taguchi using a recurrent neural network to process time-series data. The input for this procedure uses motion gestures that describe both alphabetic characters and full words in Japanese. [3]

Klimis Symeonidis expanded upon this by designing a system that could recognize American Sign Language (ASL) from still images of hands [4]. Ibraheem and Khan also developed a hand gesture system for recognizing Arabic Sign Language using both Ellman and Fully recurrent neural networks and made comparisons on each method. They also implemented a color-based image processing method to determine location of each finger and wrist in the image. [2]

3. IMAGE RECOGNITION

A color recognition algorithm is implemented to determine the position of each of the subject's hands relative to their center of mass. Colored patches are worn on both hands and in the center of the chest. The red patch represents the left hand, the blue patch represents the right hand, and the green patch represents the middle of the chest.

A standard webcam is used to take the input video. VLC Media Player is setup to take a snapshot image of the video every 1 second. The subject then records videos of themselves posing for each of the four gestures. These videos are automatically converted to images of 640 x 480 pixels. The final input set consists of 80 training images and 300 testing images. These images will be used to extract input data for the neural network.

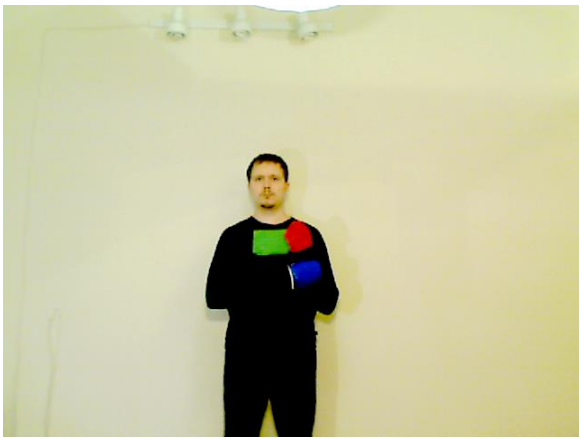

Figure 2: Example of an input image for training the holding gesture

The first step in the image processing routine is to rasterize each image into an array of red, green, and blue components. Each array is then examined pixel by pixel and the color components are compared to a preset value on a scale of 0 to 255. If the red component is greater than 80 and the green and blue components are less than 10, then the pixel is designated as a red pixel. If the green component is greater than 60 and the red and blue components are less than 20, then the pixel is designated as a green pixel. Finally, if the blue component is greater than 40 and the red and green components are less than 10, then the pixel is designated as a blue pixel. These values should be adjusted to match the accuracy of the webcam used.

Next, each pixels location in each color category is added up and divided by the total number of pixels in that category to get the average position of each color. If any color contains zero pixels, then that image is discarded. The results of the input of the training data is shown in Figures 3.1-3.4.

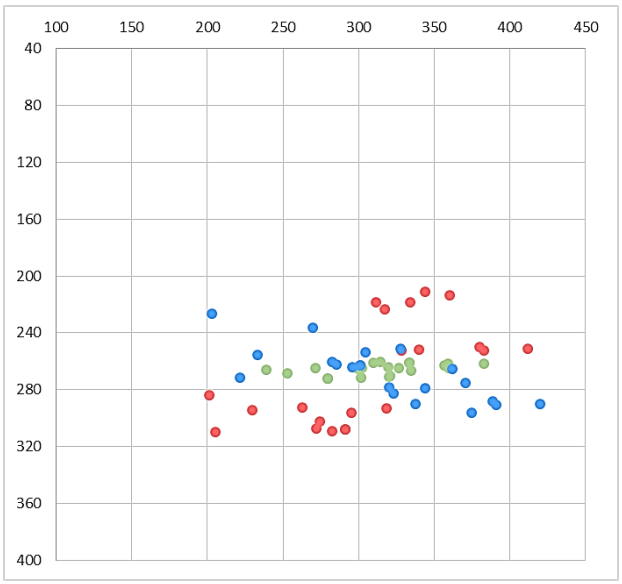

Figure 3.1: Input data for training the holding gesture

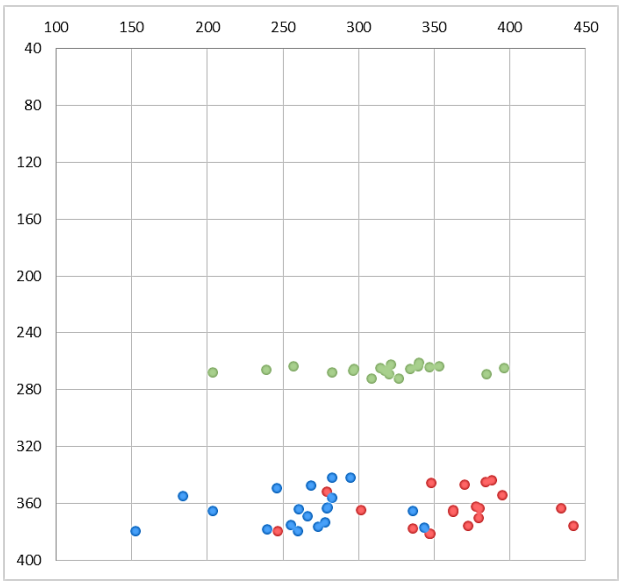

Figure 3.3: Input data for training the offside gesture

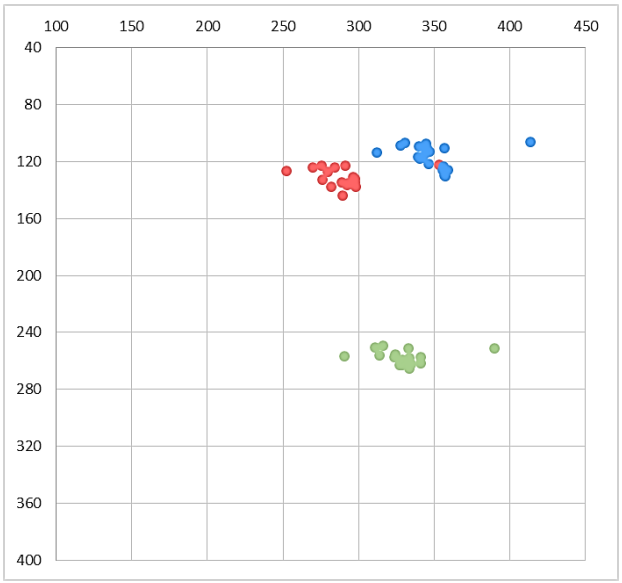

Figure 3.2: Input data for training the timeout gesture

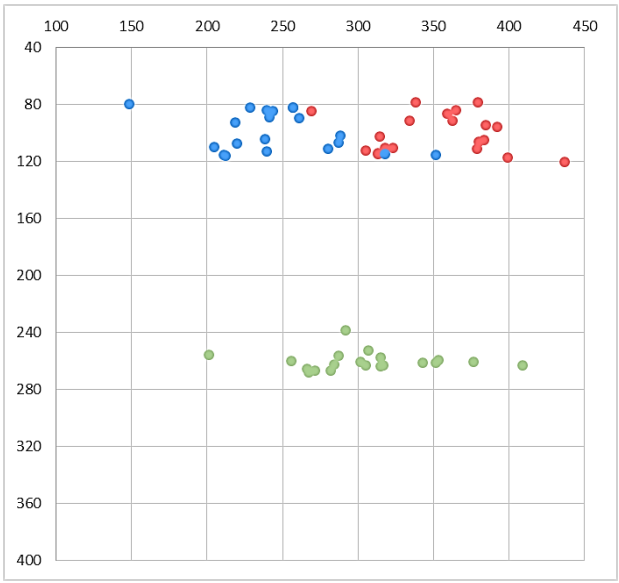

Figure 3.4: Input data for training the touchdown gesture

Finally the average position of the green pixels is subtracted from the average positions of the red, green, and blue pixels so that green is centered at (0, 0). This corrects for any horizontal or vertical movements of the subject during the input video capture process. The result of this adjustment is show in in Figures 4.1-4.4

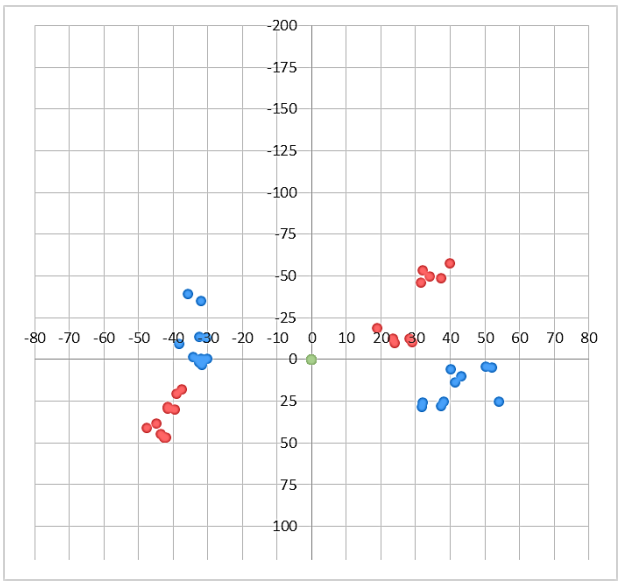

Figure 4.1: Input data for training the holding gesture after adjustment

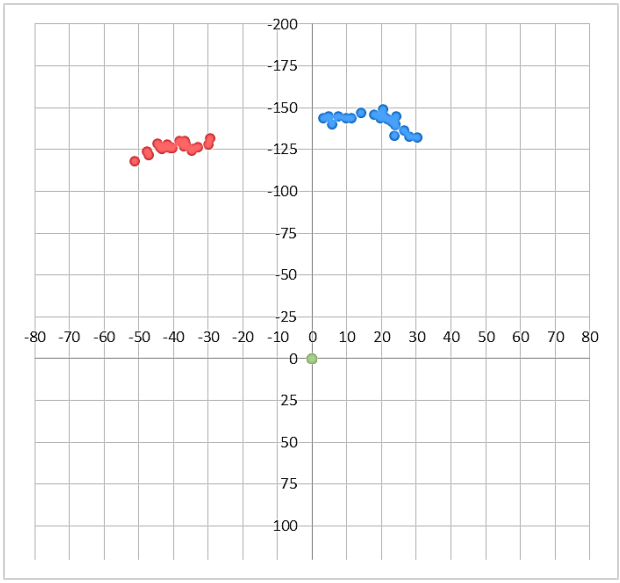

Figure 4.2: Input data for training the timeout gesture after adjustment

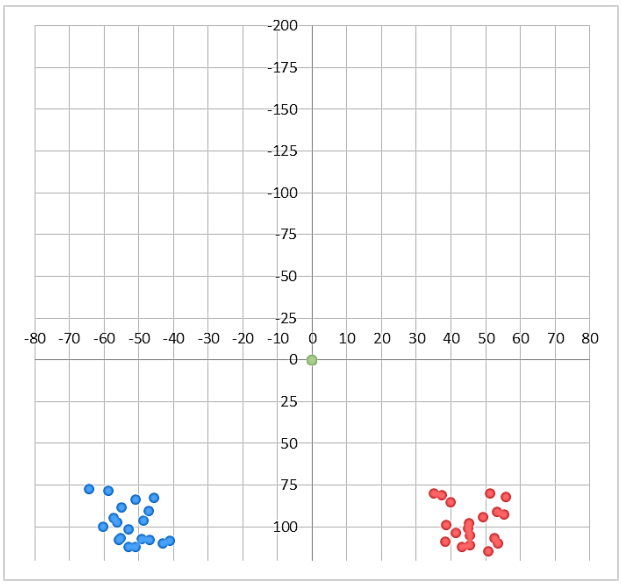

Figure 4.3: Input data for training the offside gesture after adjustment

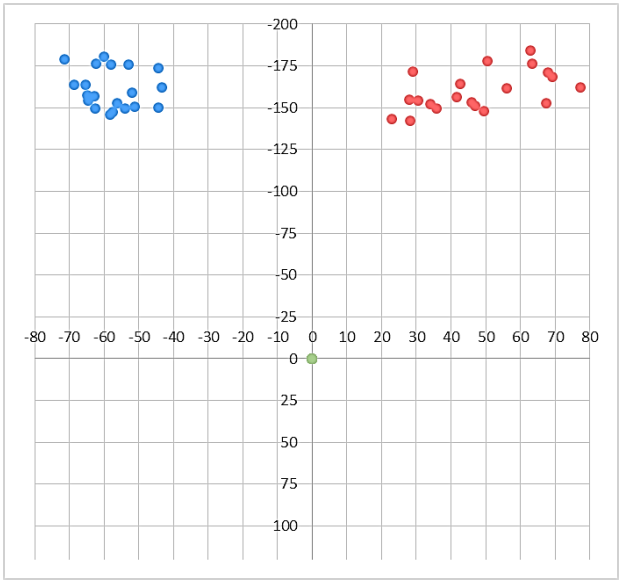

Figure 4.4: Input data for training the touchdown gesture after adjustment

The x and y components of each of the red and blue values are then used as input values to the neural network. This provides well defined clusters for which the network can create decision boundaries. The input training data is shown in Figures 5.1 and 5.2.

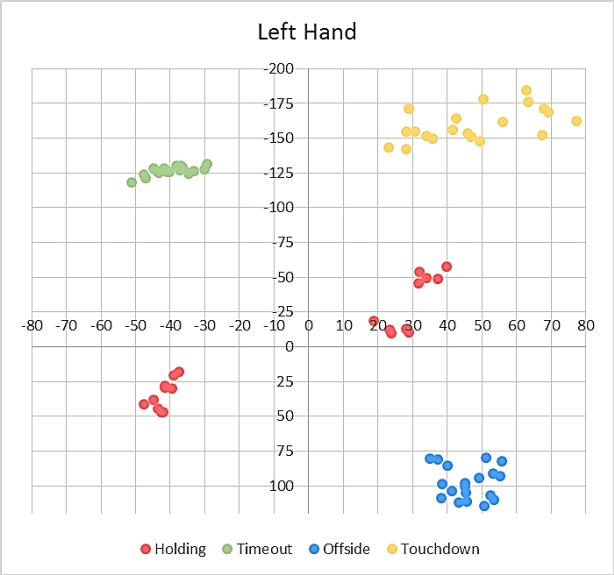

Figure 5.1: Data input into the neural network for the left hand

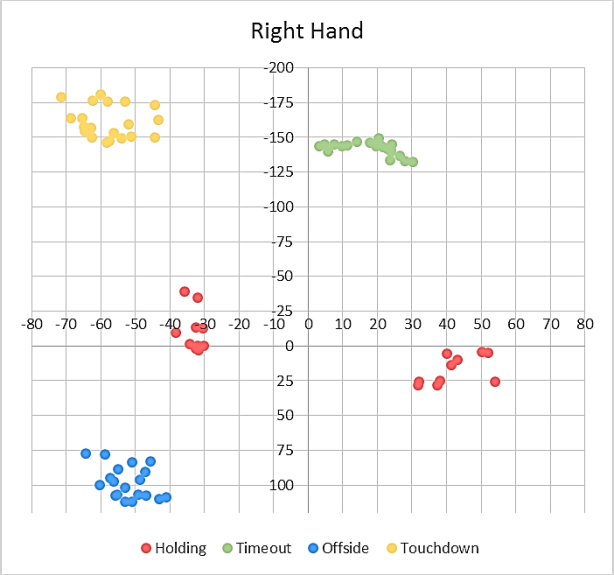

Figure 5.2: Data input into the neural network for the right hand

4. NEURAL NETWORK

A single layer neural network is trained to recognize the gestures from the input data displayed in Figures 5.1 and 5.2. A set of target data is loaded from text file specified by the user. These target values correspond to the expected output of the neural network given the associated input image.

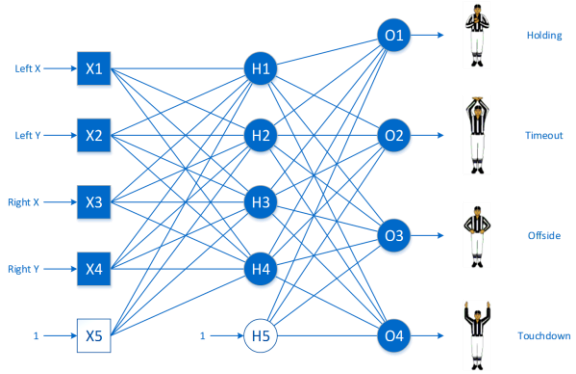

Figure 6: Diagram of neural network

The network is setup with 5 input nodes (1 bias node), 5 hidden nodes (1 bias node), and 4 output nodes as shown in Figure 6. A sigmoid activation function is used at each of the hidden and output nodes. The network was tested with 2, 4, 6, and 8 hidden nodes and it was determined that 4 hidden nodes were optimal as it gave the least error and ran in an acceptable amount of time. The results of this test are shown in Table 1.

Nodes	Training Error %	Testing Error %
2	2.6133	12.5755
4	0.0013	0.0705
6	0.0014	2.3517
8	0.0012	3.3680

Table 1: Error results from test with different number of hidden nodes

The weights of the network are initialized randomly from 0 to 1. Each hidden and output node uses a sigmoid function as its activation function. The target value for each output node is 1 for the correct gesture and 0 for incorrect gestures.

4.1. FEED-FORWARD NETWORK

Each set of input values is inserted into the input nodes x. Equation 1 is then used to calculate the output value of each hidden node h.

$$h_j = \frac{1}{1 + e^{-z_j}} \qquad z_j = w_{0j}^H + \sum_{i=1}^{I} x_i w_{ij}^H$$

Equation 1: computes the output value for each hidden node

In Equation 1, i represents the index of the input node, j represents the index of the hidden node, and w_{ij}^H is

the weight from input node x_i to hidden node h_j. Once all hidden nodes are computed, then the output values are computed for each output node o using Equation 2.

$$o_k = \frac{1}{1 + e^{-z_k}} \qquad z_k = w_{0k}^O + \sum_{j=1}^{J} h_j w_{jk}^O$$

Equation 2: computes the output value for each output node

In Equation 2, k represents the index of the output node, j represents the index of the hidden node, and w_{jk}^O is the weight from hidden node h_j to output node o_k. Each output node value is then compared to the target value t_k to give the output mean squared error using Equation 3.

$$Error = \frac{1}{K}\sum_{k=1}^{K}(t_k - o_k)^2$$

Equation 3: computes the mean squared error of all output nodes

The total error is computed over all of the training data sets by summing all errors for each input set. Once the total error has reached a minimum (not necessarily the global minimum) then the network is determined to be trained.

4.2. BACKPROPAGATION ALGORITHM

To train the network, a backpropagation algorithm is implemented. First, the error of each output node δ_k is computed using Equation 4.1. Then, the new output weight values are computed for each weight in w^O using Equation 4.2.

$$\delta_k = o_k(1 - o_k)(o_k - t_k)$$

Equation 4.1: computes the error in each output node

$$\Delta w_{jk}^O(t) = -\alpha\delta_k h_j + \beta\Delta w_{jk}^O(t-1)$$
$$w_{jk}^{O\,\prime} = w_{jk}^O + \Delta w_{jk}^O$$

Equation 4.2: computes the new output weight values

Once each weight is calculated in w^O, the error for each hidden node δ_j is calculated using Equation 5.1. Then, the new hidden weight values are computed for each weight in w^H using Equation 5.2.

$$\delta_j = h_j(1 - h_j)\sum_{k=1}^{K}\delta_k w_{jk}^O$$

Equation 5.1: computes the error in each hidden node

$$\Delta w_{ij}^H(t) = -\alpha\delta_j x_i + \beta\Delta w_{ij}^H(t-1)$$
$$w_{ij}^{H\,\prime} = w_{ij}^H + \Delta w_{ij}^H$$

Equation 5.2: computes the new hidden weight values

Once the weights have been adjusted, the next input set is taken and the network is trained this new input set. Once the algorithm has been applied to each input set, the total error (computed from the sum of errors from each input set) is then compared to the previous total error of the network. If the new total error is found to be less than the previous, then the procedure starts again. If the new total error is equal to or greater than the previous, then the network has found a minimum error state and training ends.

The α value in Equations 4.2 and 5.2 is the network's learning rate. It determines how quickly the weight values change. The network is tested 10 times with α = 0.1, 0.05, 0.01, and 0.005 and the results are show in Table 2. The optimal learning rate was determined to be 0.01 as it resulted in the lowest error rates.

alpha = 0.1			**alpha = 0.05**		
Iterations	Training Error %	Testing Error %	Iterations	Training Error %	Testing Error %
751	9.2640	41.6272	3653	2.5053	15.0205
1553	7.6639	32.1983	2443	7.5927	32.0255
1597	7.6638	32.1992	4825	2.7538	10.0151
2226	6.9146	25.6860	445	11.8314	47.3018
1559	7.6636	32.1969	1652	7.6638	32.1973
1525	7.6631	32.1949	2594	7.5928	32.0273
1353	7.6635	32.8950	2793	6.9168	25.7110
1723	7.6635	32.2142	2831	6.8455	25.5379
1429	7.6633	32.1959	3089	6.8449	25.5366
1776	7.6631	32.1948	2197	7.8434	27.2303
alpha = 0.01			**alpha = 0.005**		
Iterations	Training Error %	Testing Error %	Iterations	Training Error %	Testing Error %
6179	6.0369	27.4523	7279	7.5300	32.0858
6747	4.5448	16.2828	14367	2.2235	11.3619
165532	0.0007	0.5071	4544	6.6891	33.9608
2455	8.9953	40.1287	1615	11.5535	45.4366
5049	7.5316	32.0443	6494	5.0825	17.2966
217092	0.0012	1.8607	1705	11.5531	45.4356
239909	0.0012	0.9903	263011	0.0022	1.8782
4662	7.5331	32.0513	14294	3.2173	12.7913
11647	2.7586	10.0342	3411	10.0256	42.8244
4550	7.5328	32.3317	19271	0.9176	6.9937

Table 2: Results from tests with different learning rates

The β value in Equations 4.2 and 5.2 is the network's momentum. It is used to prevent oscillation and helps the algorithm avoid local minima. The network was tested 10 times with β = 0.9, 0.7, 0.5, and 0.3 and the result are shown in Table 3. The optimal momentum was determined to be 0.5 as it resulted in the lowest error rates.

beta = 0.9			**beta = 0.7**		
Iterations	Training Error %	Testing Error %	Iterations	Training Error %	Testing Error %
7613	2.7560	10.1678	6790	7.5258	32.3911
776	11.6824	45.6438	7981	6.7825	25.5926
1985	8.9446	40.6082	8181	6.7821	24.8493
1743	9.0458	40.1425	2968	8.9815	40.0615
4550	6.8125	25.5421	6543	7.5258	32.0576
3373	7.5594	32.0277	278459	0.0010	1.3839
3548	7.5589	32.0268	2904	8.9791	40.1723
4938	2.5079	15.0290	22659	0.9177	6.8993
2884	7.6089	32.0172	1457	11.5591	45.5284
3799	7.5590	32.0316	6172	7.5264	32.0593
beta = 0.5			**beta = 0.3**		
Iterations	Training Error %	Testing Error %	Iterations	Training Error %	Testing Error %
287227	0.0021	1.0708	7769	5.7396	21.3114
10862	2.5179	15.0637	11130	5.0321	17.4267
313823	0.0016	1.1930	9179	7.5298	32.1074
8523	5.0230	19.2297	9106	5.0290	18.7033
239154	0.0019	1.7845	5146	8.9720	40.0889
10328	6.7848	25.6835	223222	0.0028	1.3002
3911	8.9750	40.0991	6392	8.2579	28.6706
11925	5.0256	18.8486	6850	5.0333	19.0845
7971	7.5265	32.1012	243925	0.0031	1.8620
9736	5.0287	17.1043	38112	0.9217	7.4118

Table 3: Results from tests with different momentum values

The network is then tested on the test input data set which is created the same way as the training input data set, but contains far more samples. The procedure runs through each input set using Equations 1 and 2 and computes the error using Equation 3. The total error is then computed over all of the test data sets by summing the errors for each input set. This error is used to determine how well the network was trained and is examined in section 5.

5. RESULTS

The first test of the network demonstrated unpredictable results due to the network converging to local minima instead of the global minimum. To alleviate this issue, an ensemble method was implemented by training the network 10 times with different initial weights and using the result with the minimum training error. This resulted in a much more predictable outcome, although it also significantly increased the time it takes to train the network.

Run	Training Error %	Testing Error %
1	0.0013	0.3333
2	0.0016	4.1505
3	0.0013	0.9677
4	0.0019	1.2224
5	0.0022	1.1767
6	0.0018	1.4262
7	0.0014	0.8639
8	0.0018	1.3242
9	0.0022	2.8100
10	0.0017	2.3657
Avg.	**0.0017**	**1.6641**

Table 4: Results of the final test of the network

Table 4 shows the final test results of the neural network. After implementing the ensemble method, the average training error fell to almost 0, while the testing error fell to less than 2%. The maximum error obtained is 4.1505% and the minimum error obtained is 0.3333%.

An individual training loop took an average of 250,000 iterations to converge to an acceptable minimum. On a machine using an Intel Core i7-720QM CPU at 1.60GHz the full ensemble training took approximately 5 – 10 minutes to process, depending on the number of iterations in each training loop of the ensemble.

5.1. MULTILAYER COMPARISON

The network used in this experiment contains only a single hidden layer. It is possible that adding more hidden layers could increase the efficiency and accuracy of the network. To examine this, an application called BasicProp is used to simulate this network and a similar network with two hidden layers [5]. After setting up the network using the BasicProp interface, the same training data and test data is fed through each network and the resulting error is returned and shown in Table 5.

Layers	Iterations	Training Error %	Testing Error %
1	500000	5.6814	5.8693
2	500000	3.5991	14.8775

Table 5: Simulation results of multilayer networks using BasicProp

As shown in Table 5, the testing error of the network with two hidden layers is actually worse than if only one hidden layer is used. This is assumed to be a result of over-fitting of the training data. However, it was also observed that single layer network took much longer to converge to a minimum error. Further, the single layer network was also far more likely to converge to a local minimum rather than the global minimum. Due to these results, it is still determined that the single layer neural network is the optimal network due to its higher accuracy after convergence. It is also worth noting that the final error of the system developed in this experiment is much smaller than the final error of the simulated network.

6. CONCLUSION

The procedure described in this experiment contains a few weaknesses and strengths. It uses a neural network and, therefore, is prone to any disadvantage associated with neural networks. The local minima issue is the most prominent and, although it has been alleviated by implementing an ensemble method, it cannot be completely resolved. The volatility of results can make the procedure tricky to work with.

The method of creating input images can also cause issues if proper lighting is not used. The camera image quality can also become a factor. A few images could not be used as inputs because the camera could not distinguish between blue and black. While color recognition created complications, the lower resolution webcam actually improved the performance of the procedure as there were less pixels that needed to be examined.

Even with these weaknesses, the experiment also displays a few strengths. The final network is shown to be more accurate than expected, and even show less error than the algorithm used in BasicProp, even in the worst case. Once trained, the recognition process is fast and accurate, showing an average processing time of 1 millisecond per input image. Given this result, the procedure could easily be expanded to function with real-time input.

The network is also flexible enough to recognize as many gestures as the user trains it for, although the training time will increase exponentially with each added output. It is also possible to increase the number of input nodes to include elbow, head, and feet positions.

7. FUTURE WORK

Much work could be done to increase the effectiveness of the recognition software. The most prominent improvement would be to do away with color recognition for inputs and use a more sophisticated method. Further hardware could be implemented to include depth analysis and, possibly, a second neural network trained to recognize positions of hands and chest without the need for colored patches.

Another improvement would be to expand the functionality to recognize hand motion such as waving by interpolating the results of two taken in succession. In theory, it would be possible to expand functionality to become as effective as modern video game image recognition devices for use in real-time.

8. REFERENCES

[1] Harling, P. A. (1993, March 18). Gesture Input using Neural Networks. Department of Computer Science. Heslington, York, UK: University of York.

[2] Ibraheem, N. A., & Khan, R. Z. (2012). Vision Based Gesture Recognition Using Neural Networks: A Review. International Journal of Human Computer Interaction, Vol. 3 Issue 1.

[4] Murakami, K., & Taguchi H. (1991). Gesture Recognition using Recurrent Neural Networks. Kawasaki, Japan: Fujitsu Laboratories LTD.

[3] Symeonidis, K. (2000, August 23). Hand Gesture Recognition using Neural Networks. School of Electronic and Electrical Engineering. Guildford, Surrey, UK: University of Surrey.

[5] "Basic Prop: A Simple Neural Network Simulator" Available: http://basicprop.wordpress.com/2011/12/21/introducing-basic-prop-a-simple-neural-networ/. [2013, December 5]