Shaun Howard 9/23/15
EECS 484: PS4 smh150

# PS4: Maxnet

## Initialize Weights of Influence:

First, I created a function called "initialize_weights.m" to initialize the weights of influence of each neuron on every other neuron in the maxnet. I initialize the neuron weights with unity autapse and mutual inhibition. These weights determine the influence of neuron j onto neuron i. The neurons compete until a winner neuron is found. The battle happens this way in pseudocode:

```
For each node i in maxnet:
    For each selected node j in maxnet:
        If maxnet(i) == maxnet(j):
            W(i,j) = 1
        Else:
            W(i,j) = -epsilon
        End If
    End For
End For
```

Here, W is the influence weight matrix of all the neurons. We can think of this in terms of a 2-D matrix representing 2 sets of neurons. The neuron in maxnet at column j will affect the neuron in row i. The produced weight is the correspondence of influence when training the neurons in their competition phase. This phase is the key to the maxnet.

Hence, if the selected neuron j matches the neuron under inspection, the influence weight of j on i is set to 1, else the weight is negative so that it decreases the activity of other neurons in the maxnet. In this network, each neuron negatively influences all neurons besides itself, called mutual inhibition. Neurons in the maxnet also have unity autapse which means neurons have a positive influence on themselves. Also notice how my epsilon value is positive, but the influence of neuron j on i is assigned the negative value of epsilon for the purpose of competition.

The epsilon values I used varied. I realized that a good way to choose an epsilon was to have it lie in the range between 0 and 1/(number_of_neurons). The values I experimented with will be discussed in a later section of my document.

## Neuron Activation Function:

The second thing I did was finish the provided activation function. When the neuron input is provided to the activation function, it will return a positive activation value if the input is positive or 0 when the input is negative. The input corresponds a selected (ith) neuron activation value in a maxnet.

Find Winner:

I also added a function to determine if there is a winner, a.k.a. a single positively firing neuron, given the previous iteration's calculated sigma vector. This function is simply named "find_winner.m". It is used to determine when to stop the "while" loop of training. Hence, the loop is a post-condition loop, which runs at least once and plans to stop only when a single positively active neuron is found in the maxnet. It also determines that a "winner" has been found when 0 neurons have positive activation values for the purpose of ending the program, but it just so happened that the best neurons tied and died in competition.

Experiments:

In my experiments, I tweaked two values: epsilon and the number of neurons. For a given number of neurons, I tested various values of epsilon to find any patterns that might exist. I stuck with the values between 0 and 1/(number_of_neurons) because anything beyond that would be unstable in learning. The values of epsilon and the number of neurons to store in the network as well as the number of iterations taken to converge were experimented with as follows:

| Trial Number | Number of Neurons | Epsilon | Iterations to Converge |
|---|---|---|---|
| 1 | 10 | 0.005 | 250 |
| 2 | | 0.05 | 26 |
| 3 | | 0.15 | 7 |
| 4 | | 0.195 | 4 |
| 5 | 100 | 0.00025 | 3679 |
| 6 | | 0.0025 | 541 |
| 7 | | 0.0075 | 112 |
| 8 | | 0.00975 | 108 |
| 9 | 1000 | 0.000025 | 25930 |
| 10 | | 0.00025 | ~6000 |
| 11 | | 0.00075 | ~2500 |
| 12 | | 0.000975 | ~1000 |

Analysis:

I noticed that after running the network on even 100 neurons that learning took a long time. A trend I saw was that after probably 100 iterations, many of the neurons had already had an activation value of 0. There were only some best-performing competitors left in the mix at that point. These best competitors battled for a long time until one won. Running the maxnet on 1000 or more neurons took a very long time. I had to leave my computer or do other things while the learning took place.

The way I chose epsilon values was the following:

I know that range between 0 and 1/(number_of_neurons) is a good range for epsilon to be set at.

I selected the following quantiles from the best values range for epsilon: 2.5%, 25%, 75%, 97.5%

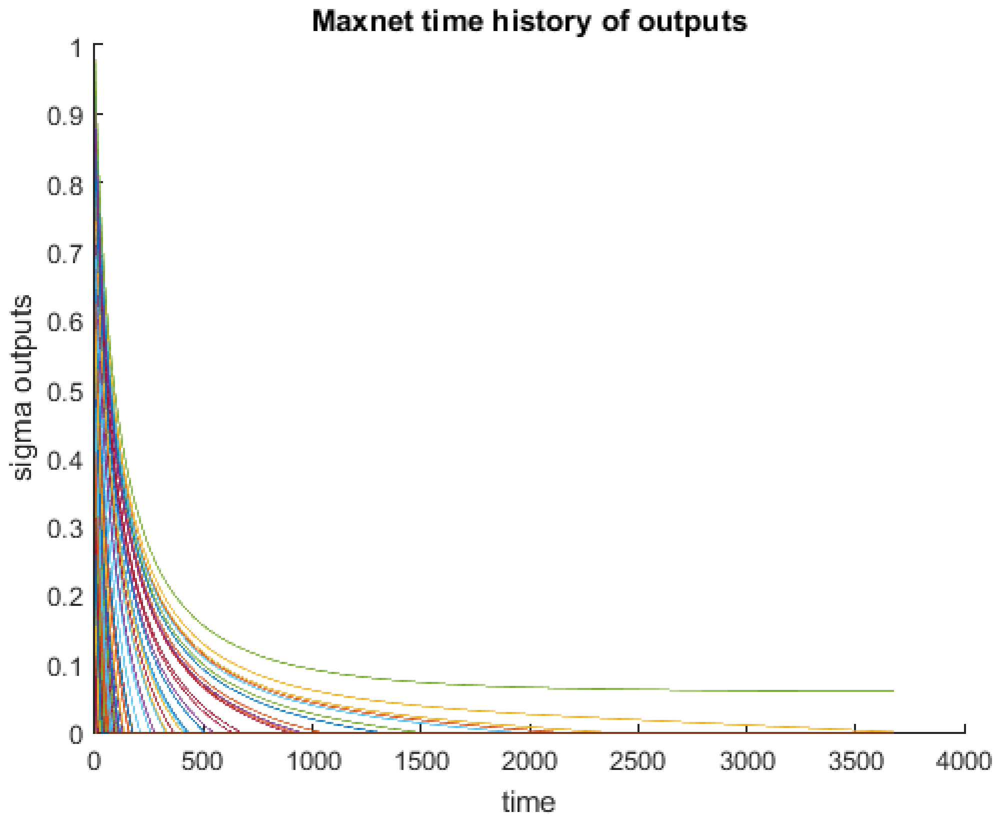
By means of analyzing the data in the table, we can tell that as the number of neurons in the network grows, the network gets more complex and makes it harder for other neurons to compete because they have more imposing competition. This means that many neurons die off in the early iterations, but as time progresses, the strongest neurons survive. This is a type of "survival of the fittest" behavior in biology. Near the end, the few best neurons "battle it out" until the best prevails. This means the neuron that prevails is the only neuron left with a positive activation energy, and it is the winner of the maxnet.

We can also tell from this data that as epsilon decreases, the time (in iterations) that it takes to find the best neuron in the network increases. This is an inversely proportional relationship. Hence, the closer the value epsilon gets to 0, the longer the neurons will compete. This reveals the purpose of the term "influence". Early on, the distributed negative influence (mutual inhibition) that neurons have on each other kill the weak neurons quick. Then, as time progresses, there are less neurons to affect the network and hence, the best neurons last the longest. These best neurons duke it out until a best has the only positive activation value by a matter of chance.

However, when epsilon is small, it takes a considerable amount of iterations to find the winner when there are 100 or more neurons. When epsilon approaches the limit 1/(number_of_neurons), the time it takes to find the winner neuron is fast relative to the number of neurons in the maxnet. Hence, if we choose an epsilon near this limit, we get relatively quick convergence of our network as compared to the number of neurons in the network.

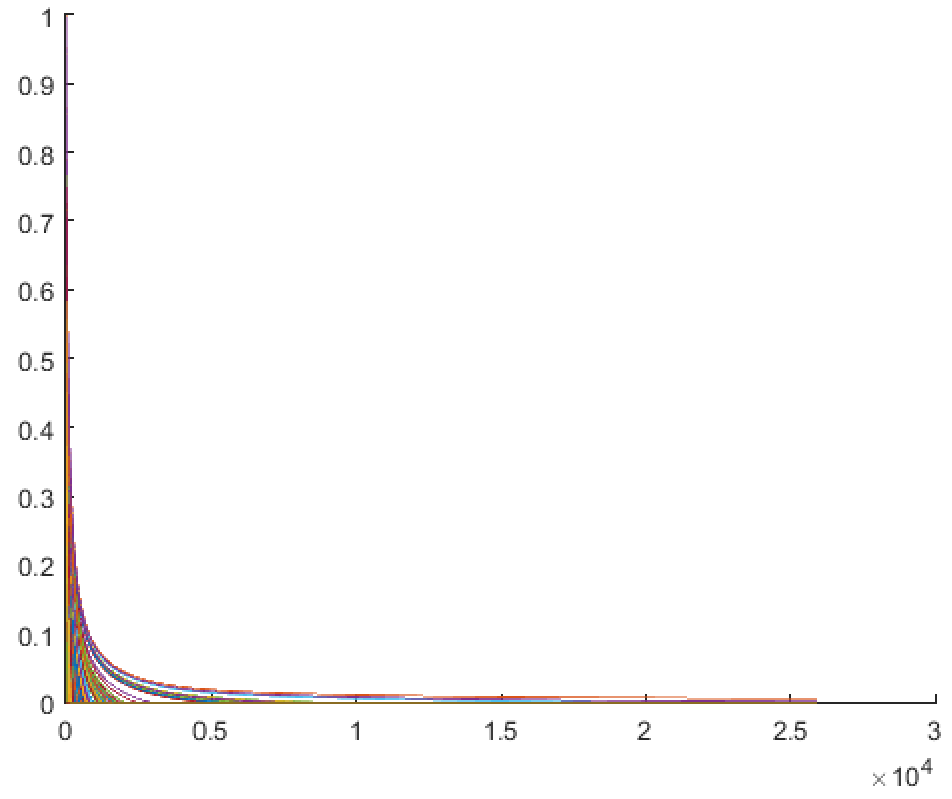
Testing on large numbers of neurons, I have noticed that a majority of the neurons in the maxnet that compete die after about 20% of the total iterations to find the winner have passed. This is noticeable in the graphs produced after running the maxnet with the given experimental values from the table above. I am not sure what this means for any specific data set, but it shows that most neurons die by 1/5 of the total time needed to find the winning neuron in the maxnet. This could be a good behavior is one is searching for a few of the best values in a neural network. The net would most likely provide a good estimate of the best neurons in the network if run on a large amount of neurons with an epsilon value corresponding to the value range I have proposed previously.

I have included a graph of experiment 5 to demonstrate the trends I have been noticing below:



As mentioned, we can see the quick convergence of most of the nodes in the network by the time 1/5 of max iterations have passed. We can also see that with time, only the fittest neurons survive in the competition network. Additionally, compared to the smaller network with 10 neurons, this network took a substantial amount of time to converge (~3,700 iterations) with a smaller epsilon as compared to this sized network with a larger epsilon near the limit of 1/(number_of_nodes), approaching from the left-hand side. 1000 neurons takes yet an even longer time.

I have also included the graph for experiment 9 below:



Again, we can notice the same trends taking place.

<u>Issues</u>:

Some notes on this process have to do with reaching a goal. Sometimes this algorithm will not reach a goal I have found. Two numbers may battle until they both go to 0. When this happens, there are simply no solutions due to the randomness of the dataset. The values that are the best, but the same, will battle until the end, when they all go to 0 or get very close to 0. I also noticed that sometimes, they will just all kill each other and no winner will be found for that reason. Lastly, I have found that some neurons will just live and their activation values will never decrement each other. Although this sounds like a bug, this happened on the code that I had run successfully on other examples. It just so happened that neurons were not being competitive enough to find a winner. I am not sure if this is a corner case of the algorithm, or if it is a bug in the code, but it happened a few times on the more complex and larger network trials, i.e. with 1000 neurons.

<u>Conclusion</u>:

We can conclude that this method of learning may take a long amount of time to find the "best" neuron. It will probably work fastest with smaller amounts of data, but it may work best by having a smaller epsilon. Having a smaller epsilon value will allow underdog neurons to potentially gain the lead over time rather than kill them quickly. This is good if there is data in the network that is nearly the same, but one value is better than the other. It will make the underdog potentially beat out the greedily-found winner as found when epsilon is larger.

We can also conclude that a good value to choose for mutual inhibition lies within the range 0 and 1/(number_of_neurons). This range was determined by experimenting with various epsilon values outside of that range and noticing that the network behavior was unstable. The neurons cannot properly compete if epsilon is too large, and if epsilon is negative, there will never be one winning neuron because all neurons will be positively influencing themselves and each other.

Finally, we can notice that the number of neurons in a network determine its complexity. For instance, if we have fewer neurons, competition is limited and the best neuron is found quickly. However, for larger networks, the competition is stiff and only the "fittest" neurons survive until eventually the best one is found. It seems that finding the best neuron takes an exponential amount of time. As analyzed previously, we notice that most of the neurons in competition in the maxnet die by the time 20% of iterations until win have passed. Subsequently, competition takes longer and longer to undergo. This reveals the exponential nature of the maxnet system.

In the end, we can conclude that if a problem calls to find the "best" neuron of all and time is not an issue, then a maxnet may be the proper solution. Also, if one is determined to find the n best neurons of the network, then they can limit the number of iterations to match when there are n best neurons (n neurons with positive activation values). This may be helpful in cases where the "best" is not completely necessary, but the top competitors are valuable. Lastly, if someone needs to make inferences about multiple data types or needs a memory efficient way to compare types of data, then the maxnet probably is not for them.