Function: state vector & inhibitory weights

A function was created to create both the state vector, and the inhibitory weights of one neuron with other neurons. This function receives the inputs of dimensionality, length constant, and maximum inhibition. The state vector was created through a simple loop, which turned neurons if between 20-60 into 40s, and 10 otherwise. Next, to calculate the inhibition weight of neuron 1 relative to every other neuron including itself, I created a loop from 1 to half of dimensionality (1 to 40 in our case), and programmed the following calculations for weight:

- Max_strength * exp{-distance_between_i_and_j / length_constant}

This equation used means that the inhibition gets weaker as it gets close to neuron 40. Essentially, here we are creating a vector of the amount of inhibition neuron 1 is inhibiting/being inhibited by other neurons.

Similarly, from half dimension to dimensionality (41 to 80), the same calculations were calculated, except the inhibition was made stronger from neuron 41 to 80 (as opposed to getting weaker from 1 to 40). This was necessary because we are assuming a circle for the neurons, in which neuron 80 is adjacent to neuron 1. Thus, if we consider the inhibition from neuron 1, neuron 41 is the furthest away from neuron 1, thus the weakest relationship of inhibition. On the other hand, neuron 80 is inhibited pretty strongly by neuron 1 because it is assumed to be right next to neuron 1.

In this same function, once we get a vector the size of the dimensionality, I had to create this inhibitory weight for every neuron, not just neuron 1. To accomplish this for neuron 2, I simply had to shift every number stored in the inhibitory weight vector to the right (the last number in the vector would become the first number in the vector, the first number became the second number, and so on). This created the inhibitory weight vector for neuron 2. Through loop, this process was repeated until I got the inhibitory weight vector of all 80 neurons, which were all put in a single matrix, that was size 80x80 (column: neurons, rows = inhibitory weights of that neuron with other neurons).

Function: Calculating New State Vector

This function calculates the final state of the vector after a certain interval, which shows the full effect of lateral inhibition over time. This function receives the input of dimensionality, number of iterations, epsilon, upper limit, state vector, and inhibitory weight.

There was once big for-loop, for the number of iterations. The number of iterations correlates with the number of time (t) has passed. Within this outer loop was an inner loop that ran as many times as the dimensionality. This inner loop calculated the new state vector for all neurons. The following equation was used to calculate the new state of each neuron.

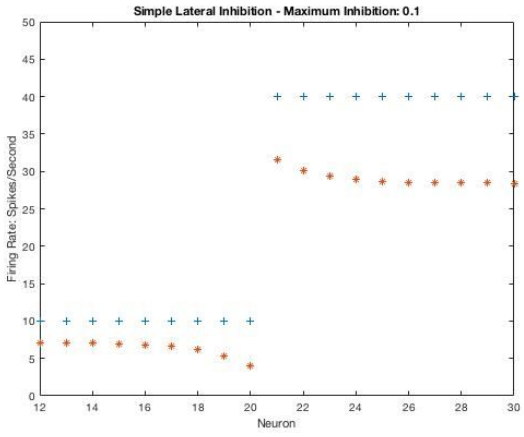
$$f[i](t+1) = f[i](t) + \varepsilon \Delta f[i]$$

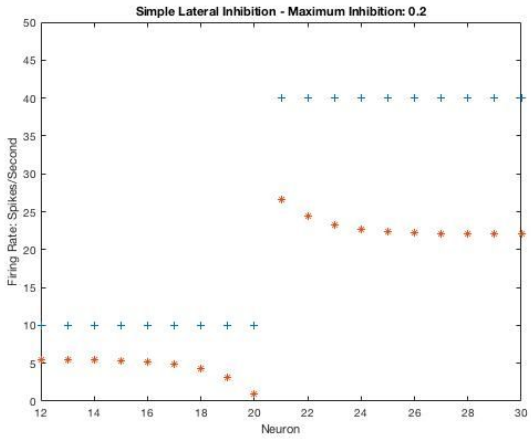
The final equation f[i](t+1) simply means the new state after one iteration, or one updating of the state vector. f[i](t) is simply the current magnitude of the neuron. epsilon*change in f[i] means the initial state of the neuron plus the dot product of inhibitory weight of this particular neuron and the states of all the neurons minus the current state, all multiplied by the constant epsilon (which regulates the amount of magnitude in change after a unit of time). Through a loop, the f[i](t+1) was calculated for every neuron

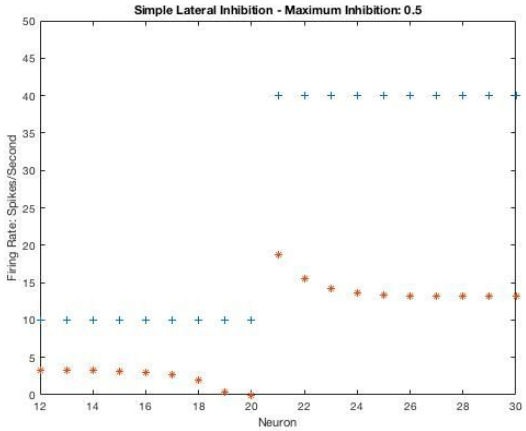
After each iteration, I made sure the lower and upper limits were not passed. This means if the new state was below 0, the value was set to 0. If the new state was above some upper limit, 60 in our case, the new state would be converted to 60. Finally, this loop is finished and the next iteration was ran. The final output was an updated state vector, which is the new state after each neuron's inhibition took its full effect on each other.

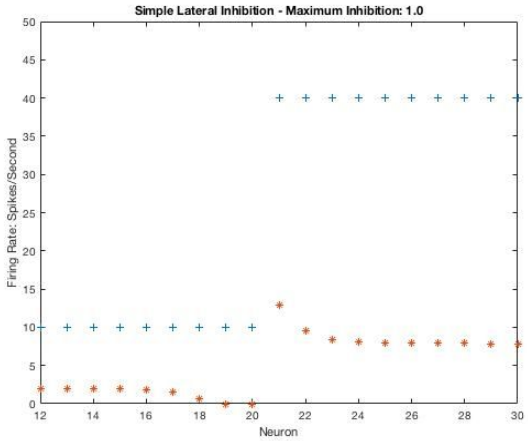
Figures:

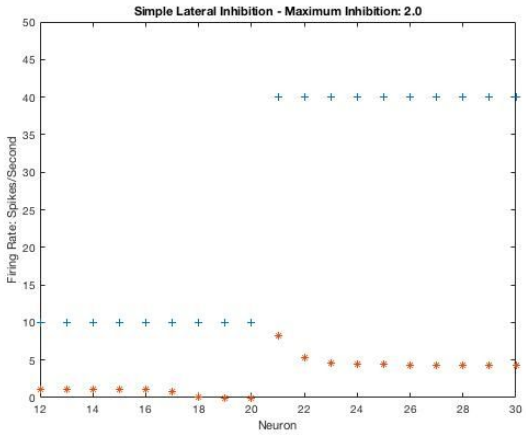
For the first 5 figures, I first initialized Dimensions, 80, iteration, 50, upper bound, 60, lower bound, 0, Epsilon, 0.1, Length Constant, 2. The only difference was the maximum inhibition, which was altered to be 0.1, 0.2, 0.5, 1.0, 2.0 respectively. In each figure, the two functions described above were used to calculate the final state, which was compared with the original state. The following is the results I got using the plot function on matlab.





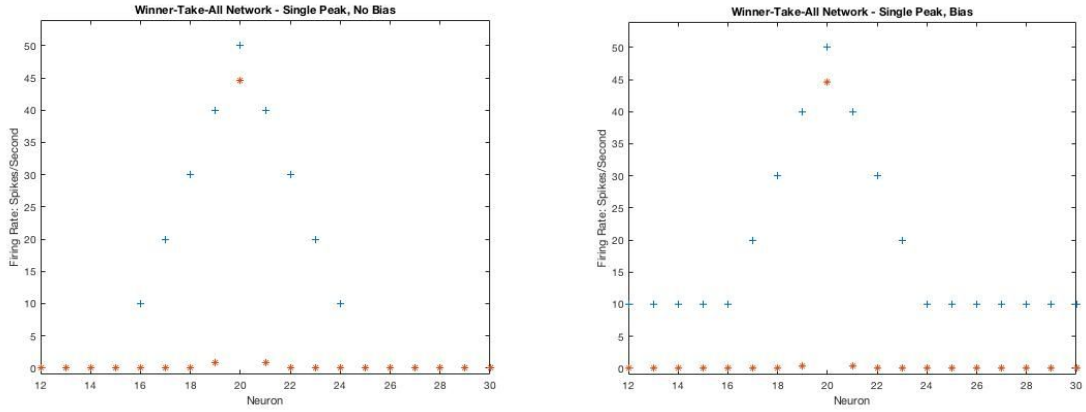






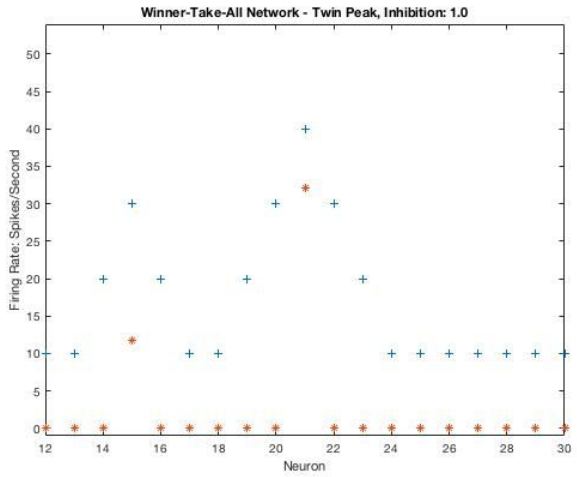
The y-axis is the firing rate, the x-axis is the neuron. Neuron is from 12-30 to stay consistent with the textbook, and to look closely at the cut-off points (where the original firing speed went from 10 to 40). The blue + sign indicates what the original state of the vector was, the * sign indicates the final state of the vectors after a certain time, in which the lateral inhibitions took full effect. The greater the maximum inhibition, the lower the firing rate became. This is expected, as greater maximum inhibition simply means that the neurons are inhibiting each other at a greater magnitude. We can clearly see that at cut off points (i.e. at 20/21), there is a biggest change in direction. At 20, the neuron is greatly inhibited by the nearest neuron neighbors to the right, which have much bigger firing rate, which leads to greater inhibiting. On the other hand, neuron 21 is the least inhibited as its closest neighbors to the left have weak firing rate, which means its inhibition effect is much weaker on neuron 21.

Next, I created a winner-takes-all (WTA) network with a single peak. Essentially, there is a clear peak (one neuron with clearly the highest firing rate) in the original vector, and after the inhibitions, there should be one clear winner at the peak. This is because the peak should inhibit nearby and all other neurons, which should lead to a single neuron dominating. The length constant became 10, which is much bigger, in order to ensure that the effect of the inhibition reaches farther and stronger, leading to a single winner. The maximum inhibition was set at 1.0 to ensure that there is enough strength in the inhibition from the peak to suppress other neurons. Self inhibition was also gotten rid of, which meant that in the inhibitory weight vector, the inhibitory weight became 0 when calculating how much a neuron is inhibiting itself. This WTA network was either calculated with all the non-peak neurons to be either 0 (no bias light level) or 10 (bias light level). The following is the graph (left is no bias, right is bias)

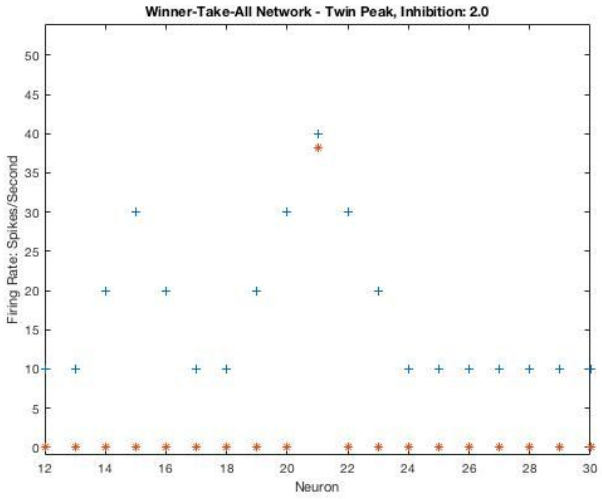


Clearly, whether there is no bias or bias, the winner take all method works. The final state vector, shown at neuron 20 in red '*' is the clear winner in terms of firing rate. With a long enough length constant, a high maximum inhibition, the peak always leads to a win as its inhibitory power is strengthened. This speaks to the power of inhibition, and the effect the parameters have on creating different types of desired outputs.

Next, this peak is made more complicated by adding two peaks, one peak bigger than the other. This two-peak WTA was tested with the same parameters as the single peak - bias WTA network. The following is the result:



Clearly, there are two neurons firing at non-zero. One clearly stronger than the other, but we want one clear winner (hence the name Winner-Take-All). The clear peak does not have enough influence on the other peak to inhibit it to zero. The following was tested with the same parameters, except the inhibition was powered up from 1.0 to 2.0



Here, clearly, there is one peak and the other smaller peak has been inhibited. This speaks to the important of the maximum inhibition. Simply by increasing how much the neurons can inhibit, the peak exerts so much power that the other smaller peak is completely inhibited.