Neural Nets & Gaussian Soft SVM Report

Christian Eid

December 2021

1. Linearly Separable Data

The goal of these machine learning algorithms is to correctly classify data. The data we want to classify consists of a vector with two values

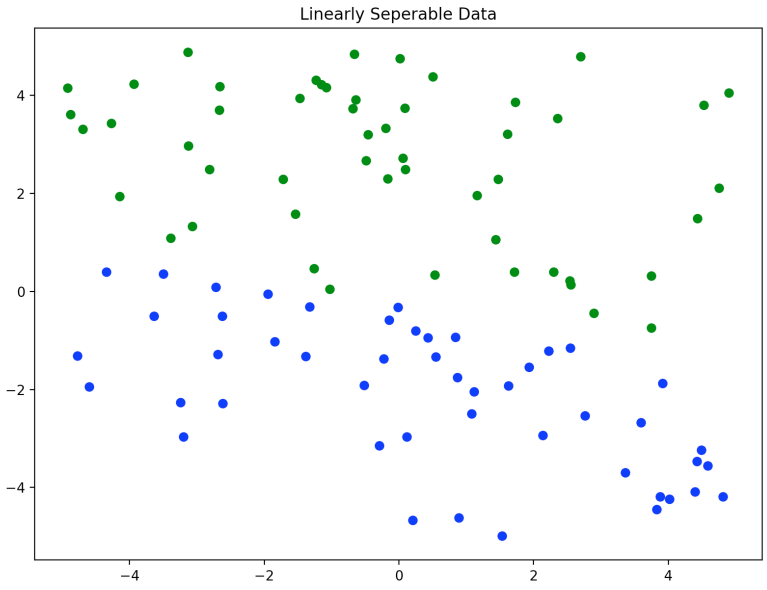
$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

We can generate a random weight vector

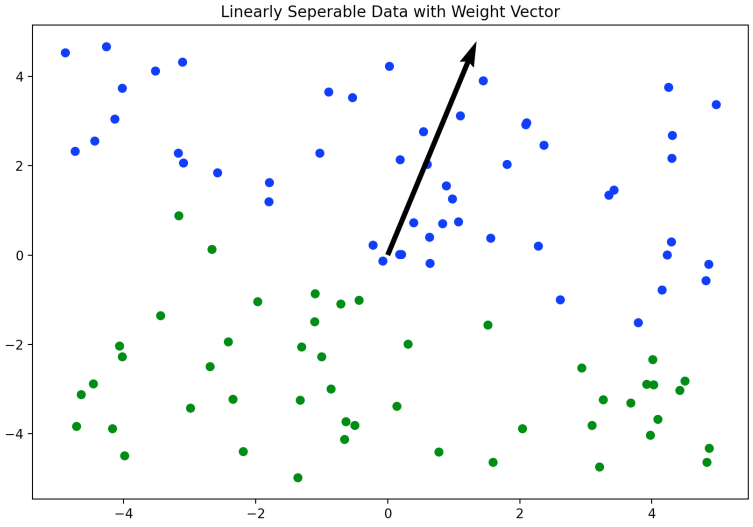
$$w = \begin{bmatrix} b \\ w_1 \\ w_2 \end{bmatrix}$$

By adding a 1 in x such that $x = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$, we can do the dot product of x and w. If that dot product is greater than 0, then we classify the data as a 1. If it is less than 0, we classify it as a 0. (Or as -1, depending on the algorithm we use to learn.) The label is y. So, if $x^T w > 0$ then $y = 1$ By doing this for every training data, we have created a set of linearly separable data. The goal of our algorithms is to learn a weight vector w that correctly classifies the data.

If we were to plot the x values using x_1 and x_2 as our axes, and color the point depending on the weight vector classification, we would obtain a graph that looks like the the following.



If we were to plot the weight vector that correctly classifies the data, it would show us that the line perpendicular to that weight vector will classify the data.



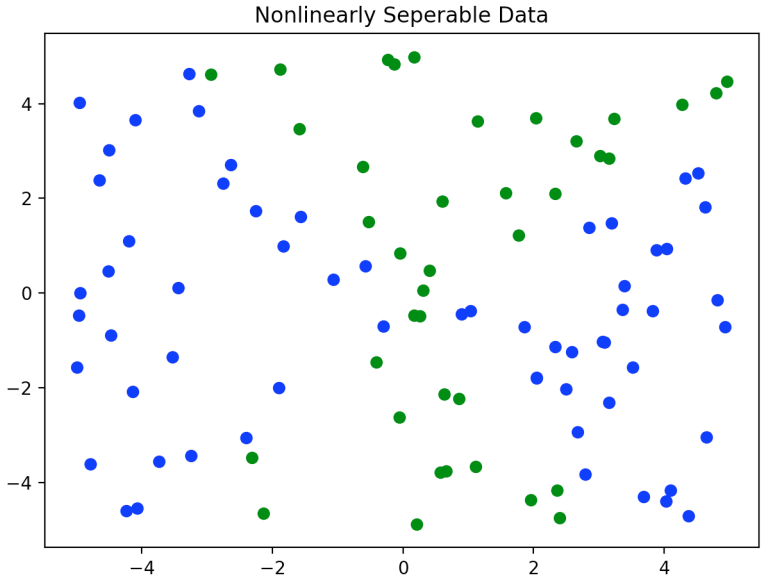
2. Non-linearly Separable Data

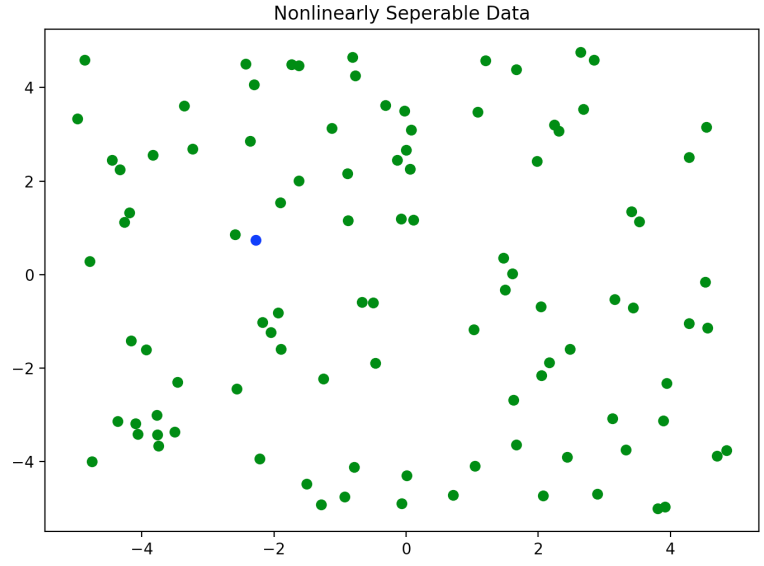
We can map each instance to a feature space of degree 2, and use it to generate a set of non-linearly separable training data. We map each value of x to a feature space such that

$$x = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_1^2 \\ x_2^2 \\ x_1 x_2 \end{bmatrix}$$

We can then use a longer weight vector $w = \begin{bmatrix} b \\ w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \end{bmatrix}$ to classify the data such that $y = 1$ when $x^T w > 0$ and 0 otherwise. This creates a set of data that is non linearly separable. Shown below are two examples of a non-linearly separable

dataset.





Note that these two could not be separated by a line, but could be separated by a polynomial degree 2 weight vector that describes a hyperbola and ellipse.

The goal of our new algorithms is to find that weight vector. They will map the instances to a feature space, and use that feature space to create a higher dimensional set of data that is linearly separable. We can then use our linear classification techniques on that feature space to generate an appropriate weight vector.

2. Neural Nets

Neural nets consists of layers of connected nodes that can classify data. The artificial neural nets we use in this program have an input layer, 2 hidden layers, and a single output node which classifies the data as either a 0 or a 1. There are weights and biases that connected each node from a previous layer to a node in the next layer. Forward propagation consists of passing a training instance through the neural net and getting an ouptut. Backpropogation calculates the loss for that training instance, and uses the loss to update the weights in the neural net one layer at a time. In this program, we use the sigmoid activation function to calculate the output of each layer, as well as the ReLU activation function to calculate the output for the two hidden layers. So, we run tests on two different neural nets, one with hidden layer sigmoid activation and sigmoid output, and one with hidden layer ReLU activation and sigmoid output.

3. Loss

We can measure the success of our algorithm's weight vector by using a loss function, which uses the predicted value of y and the actual value to calculate a loss. The negative gradient of the loss function tells us how to change our weight vector to better classify the data, since the negative gradient points to the lowest loss.

In this program, I use the Gaussian Kernelized hinge loss for the Soft-SVM gradient descent algorithm. For neural nets, the sigmoid activation function for the output lets us easily calculate the neural nets total loss. Then, we can backpropogate that loss to update the weights.

4. Soft-SVM

The Soft-SVM algorithm is a gradient descent algorithm that uses the hinge loss. It generates a random weight vector, and uses the negative gradient of a hinge loss to gradually change the value of the weight vector until the loss is minimized. The Soft-SVM algorithm has a step-size of 10 for all tests in the program. This dramatically increases the time and success rate using a step-size of 10. It also important to note that the regularization term is very low, as it would not succeed in classifying the data if the regularization was close to 1.

5. Kernels

Kernels allow us to learn to classify nonlinear data without mapping to a feature space. This allows us to learn complex non-linear weight vectors that would impossible using feature mapping. Feature mapping gets exponentially more costly when the feature space is higher dimensional. Kernels allow us to bypass feature mapping and use the original training set to classify data. Instead of using the feature space instances, it uses a special kernel function and computes the inner product between pairs of data. In this program, we use the Gaussian kernel function to kernelize the Soft-SVM algorithm.

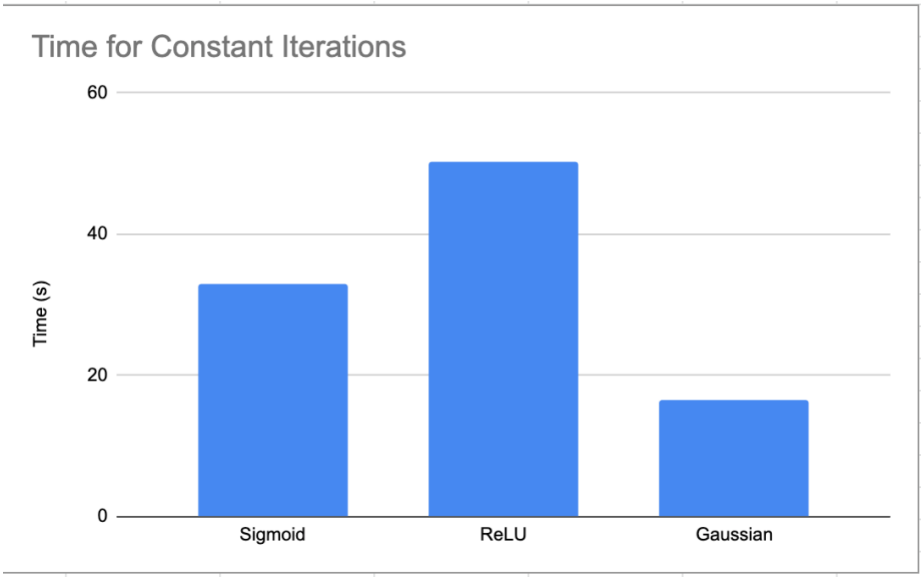
6. Data

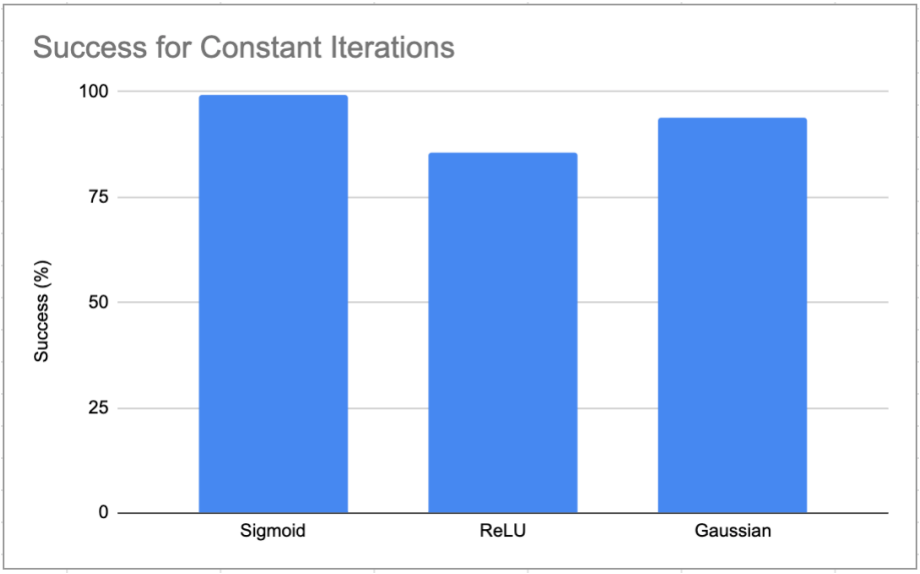
Test 1: Success and Time with constant iteration

In this test, I ran the algorithms with a constant iteration rate. For the Soft-SVM, it calculates the kernelized loss and udpates the alpha vector 15 times with a step-size of 10. For the neural nets, they use a hidden layer count of 4, and have a step-size of 0.01. They run through the 100 training instances a total of 1000 times. This is all repeated 100 times.

Algorithm	Success Rate (%)	Time (s)
Neural Net Sigmoid	99.4	32.99311697
Neural Net ReLU	85.52	50.30178316
Soft-SVM	93.71	16.51351239

At a constant iteration count, it seems that the ReLU hidden layer has a lower success rate and higher time than the sigmoid activation function. Both the neural net functions take a longer time than the Soft-SVM algorithm, but they go through the data a lot more times. The data is represented in graphs below.





Note that the neural nets are going through each training set 1000 times, which is why it takes so long. If it only goes through the training set 100 times, the following are the average success rates and time rate of the neural nets:

Algorithm	Success Rate (%)	Time (s)
Neural Net Sigmoid	85.4	2.845943175
Neural Net ReLU	76.7	4.583341574

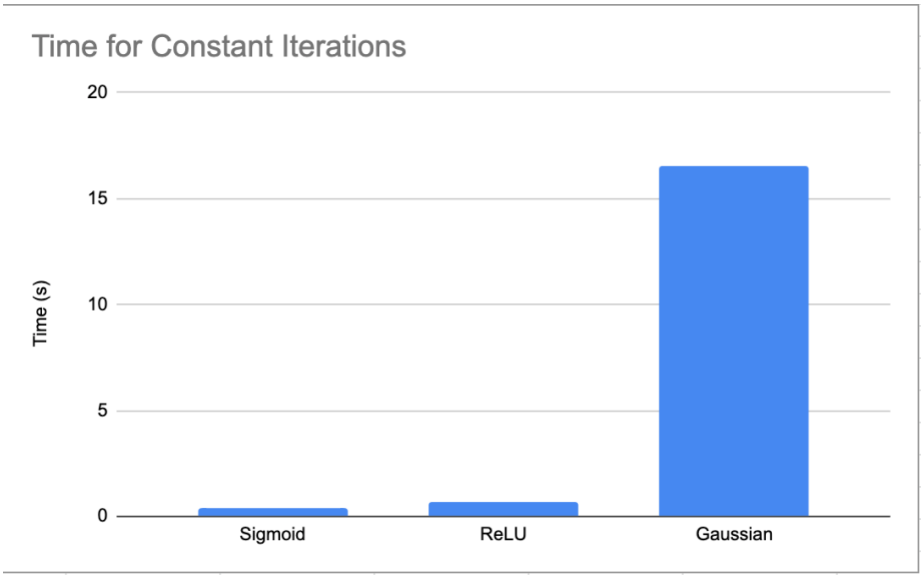
And if it only goes through the training set once, we get:

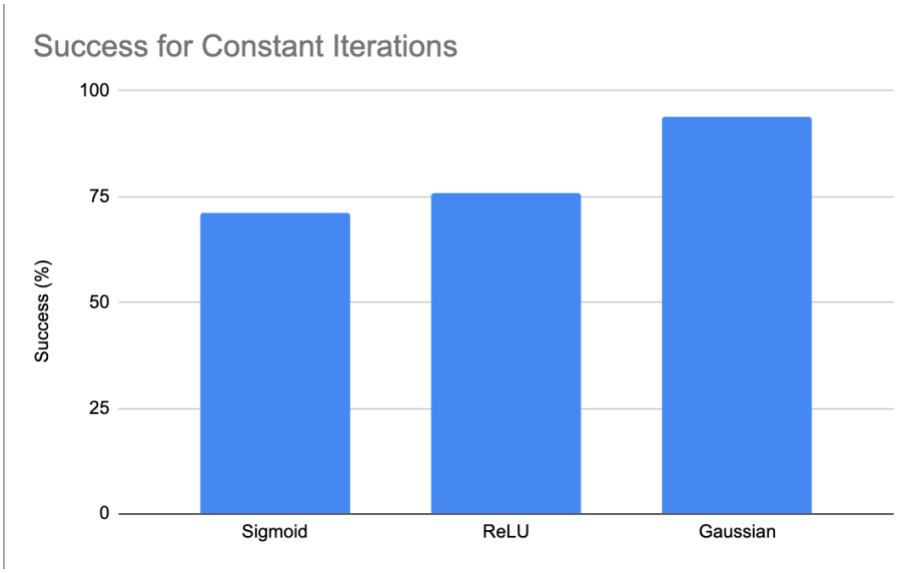
Algorithm	Success Rate (%)	Time (s)
Neural Net Sigmoid	71.2	0.0757890995
Neural Net ReLU	75.9	0.1890795635

To get this moderate success rate, the neural nets take very little time compared to the Gaussian Soft-SVM. We can compare the average of going through the data a total of 10 times to the SVM.

Algorithm	Success Rate (%)	Time (s)
Neural Net Sigmoid	76.7	0.41977563549999986
Neural Net ReLU	88.7	0.6733592569000001
Soft-SVM	93.71	16.51351239

Going through the data only once, we see that the neural nets take significantly less time than the Gaussian Kernelized Soft SVM.





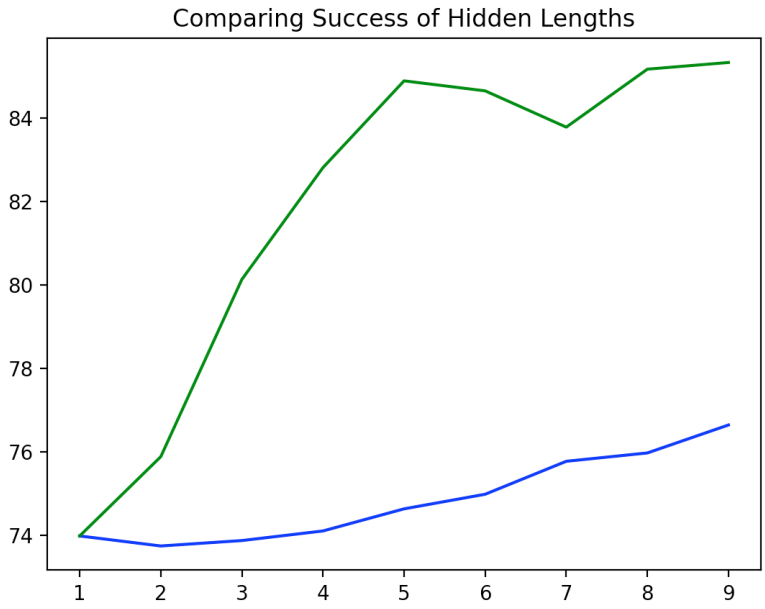
So, neural nets seem to be faster at getting a moderate success rate than the Gaussian soft-SVM.

Test 2: Hidden Length

In this test, we compare the success rate of neural nets with different hidden lengths. We create a set of data and run the two neural net algorithms on it. We create a new neural net with a hidden layer length of 1 through 9. The neural net will go through each instance in the training data once, and try to learn to classify the data. We repeat this 100 times with different sets of training data, and average out the success rate. If we plot the success rate by hidden layer length, we can see the following graph:

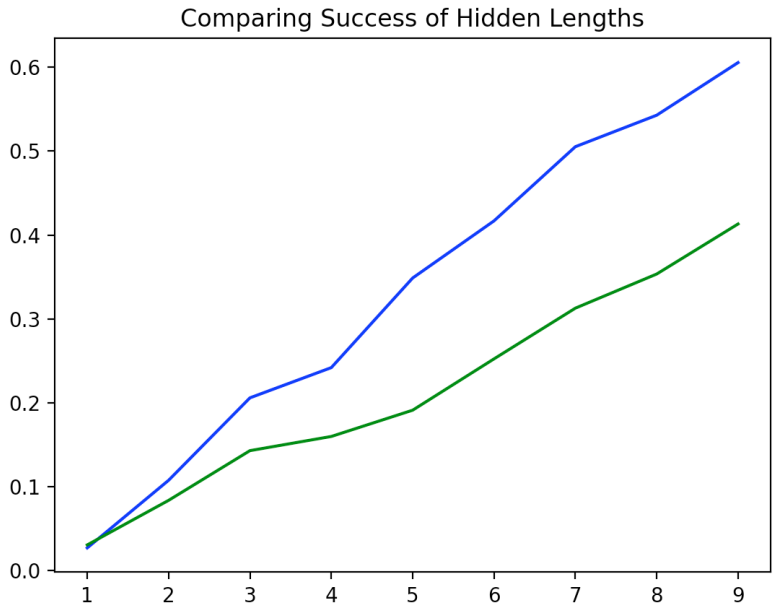
Here, blue is the ReLU neural net, and green is the sigmoid neural net. Note that the x-axis represents the number of units in the hidden layers, and

the y-axis represents the success rate at predicting the training data.



We can see that from learning by going through the data once, the sigmoid has a better shot at learning the data. Both success rates go up with the number of nodes in the hidden layers, however the sigmoid seems to learn better by going through the data once than the ReLU. Interestingly, with the sigmoid activation function, there seems to be a decrease in efficiency at 7 length hidden layer.
If we plot the time that the algorithm takes to run for different hidden lengths,

we can see an increase in the time for the number of hidden units.



For the same number of hidden units, ReLU takes on average more time than the sigmoid neural net.

7. Conclusion

The neural nets and the Gaussian Kernelized Soft-SVM are both capable of learning to classify non-linearly seperable data. In this program, however, they do not consistently hit 100% success rates. If we let them go for a while, they will generally increase in success rate to a certain point, but it will take significantly more time. The neural net is capable of learning to classify the data quicker than the gaussian soft-SVM. For all tests, the sigmoid neural net seems to be more efficient than the ReLU neural net for the way it is implemented. They both increase in their ability to succesfully classify the data when we increase the units for the hidden layers, however there is also a consistent increase in the amount of time the algorithm takes to complete.