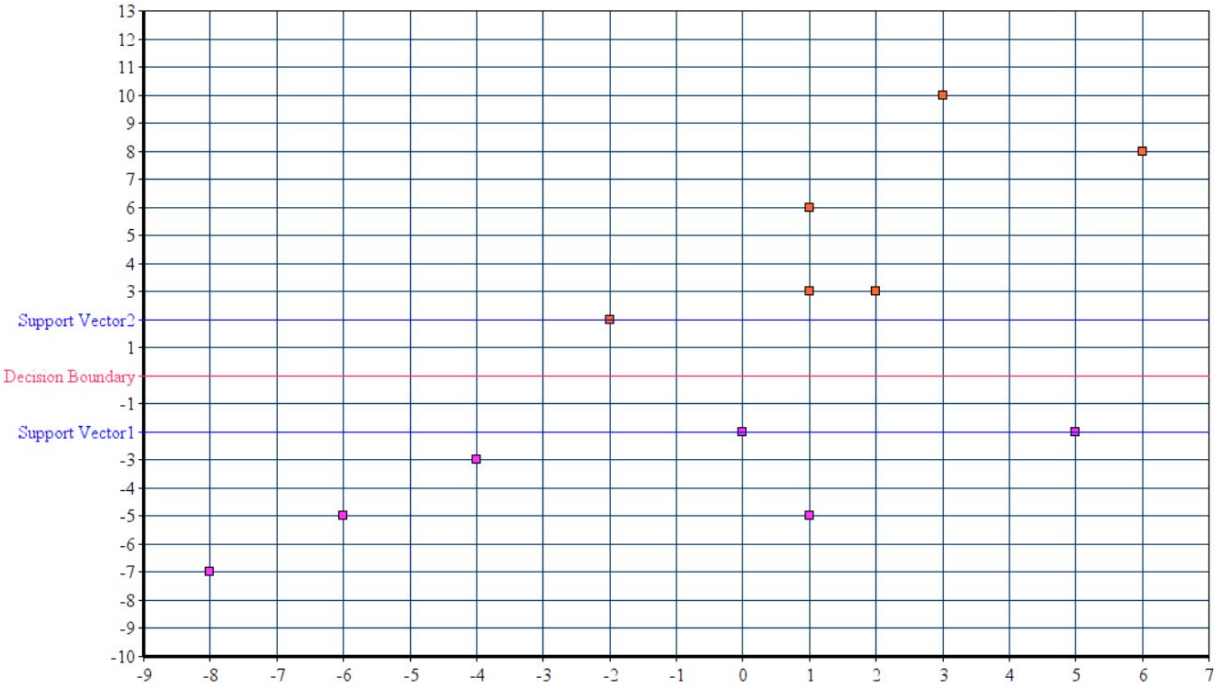
Pawel Linek, Jesse Hazard
12/1/19
SVM

Initially, we had implemented a Neural Network using a Softmax function for our classifier which performed quite well compared to the expected prediction of 20-40% accuracy with the SVM. Our neural network code worked with an accuracy of 40-60%, but based off of the assignment description, we scratched it and started working on a more “SVM” implementation. We started using the SVM-like method where when wx != y then we run the following code Max(0,wx-y+c), where c is a predefined constant for the gap between the lines.

Essentially, if we find that the decision boundary is the maximum margin classifier, meaning it’s as centered between all of our clusters of data, we will start drawing gaps between that line and the clusters. The points inside of the support vectors are more unknown as, compared to the points on the outside of the support vectors where we can more easily classify the images. So if we have something inside the support vectors(inside the gaps) we will have to guess based on which gap it is in. For example:

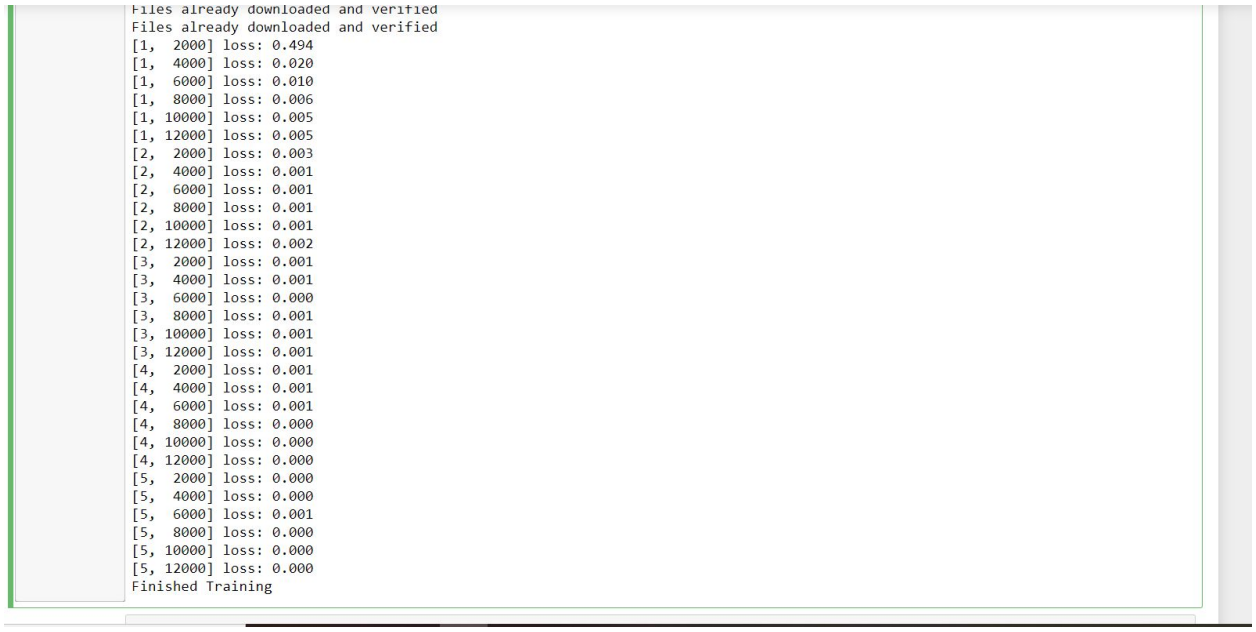


If we were to have a basic 2 class, classifier like this where the decision boundary is the red line. Then given the data points provided above, our classifier will determine the Support vectors as the blue lines. The support vectors will be the vectors we care about. If we have points on it or in the respective directions (support vector 2 will be at y=2 or greater, and support vector 1 will be at y=-2 or lesser) then we can classify those points as what class we have below, if we were to have a point in between the 2 classifiers we will have to make a probabilistic guess based on where that point lies in terms of the decision boundary. Between

Decision Boundary and Support Vector2? Most likely to be an orange point and will classify as such. The issue with our support vector machine is that sometimes one datapoint will look like the other while being a completely different class. For example if these points represented a dog or cat classifier, if we were to have a cat that looks like a dog, or a dog that looks like a cat, they will be placed in between the support vectors and can be incorrectly classified. That's why in our classifier, we have lower accuracy for similar objects, such as ship and plane. These 2 are often confused by the classifier, which makes sense as they look similar in real life and therefore, in the images. Due to an unknown reason, we have 0% accuracy on most of our classes however, the first 4 classes have at least 1% accuracy. This is our accuracy overall and for each image:

```
Predicted:   bird plane   cat   car
Accuracy of the network on the 10000 test images: 8 %
Accuracy of plane : 32 %
Accuracy of   car : 32 %
Accuracy of  bird : 15 %
Accuracy of   cat :  4 %
Accuracy of  deer :  0 %
Accuracy of   dog :  0 %
Accuracy of  frog :  0 %
Accuracy of horse :  0 %
Accuracy of  ship :  0 %
Accuracy of truck :  0 %
```

Similarly, our loss each epoch which looks at 2000 data points in a mini-batch approaches 0 after each epoch. We aren't sure why this is, but the classifier seems to focus on 1 or 2 classes when learning, as seen by the accuracy above. Our overall accuracy is 6-12% based on how we do our epochs. We believe that whatever images the classifier sees early on in it's training becomes a sort of default because, for example, there's a lot of images with the sky and if early on it learns that the sky is associated with planes, it'll start naming those as such. This can be a cause for the 0% accuracy on some of our classes.



```
Files already downloaded and verified
Files already downloaded and verified
[1,  2000] loss: 0.494
[1,  4000] loss: 0.020
[1,  6000] loss: 0.010
[1,  8000] loss: 0.006
[1, 10000] loss: 0.005
[1, 12000] loss: 0.005
[2,  2000] loss: 0.003
[2,  4000] loss: 0.001
[2,  6000] loss: 0.001
[2,  8000] loss: 0.001
[2, 10000] loss: 0.001
[2, 12000] loss: 0.002
[3,  2000] loss: 0.001
[3,  4000] loss: 0.001
[3,  6000] loss: 0.000
[3,  8000] loss: 0.001
[3, 10000] loss: 0.001
[3, 12000] loss: 0.001
[4,  2000] loss: 0.001
[4,  4000] loss: 0.001
[4,  6000] loss: 0.001
[4,  8000] loss: 0.000
[4, 10000] loss: 0.000
[4, 12000] loss: 0.000
[5,  2000] loss: 0.000
[5,  4000] loss: 0.000
[5,  6000] loss: 0.001
[5,  8000] loss: 0.000
[5, 10000] loss: 0.000
[5, 12000] loss: 0.000
Finished Training
```

One thing we could potentially do to improve our accuracy is to change the data form into higher dimensions in hopes of splitting the data up better and getting a better decision boundary. Additionally, our gap determining algorithm may not be the most accurate and could result in lower accuracy.