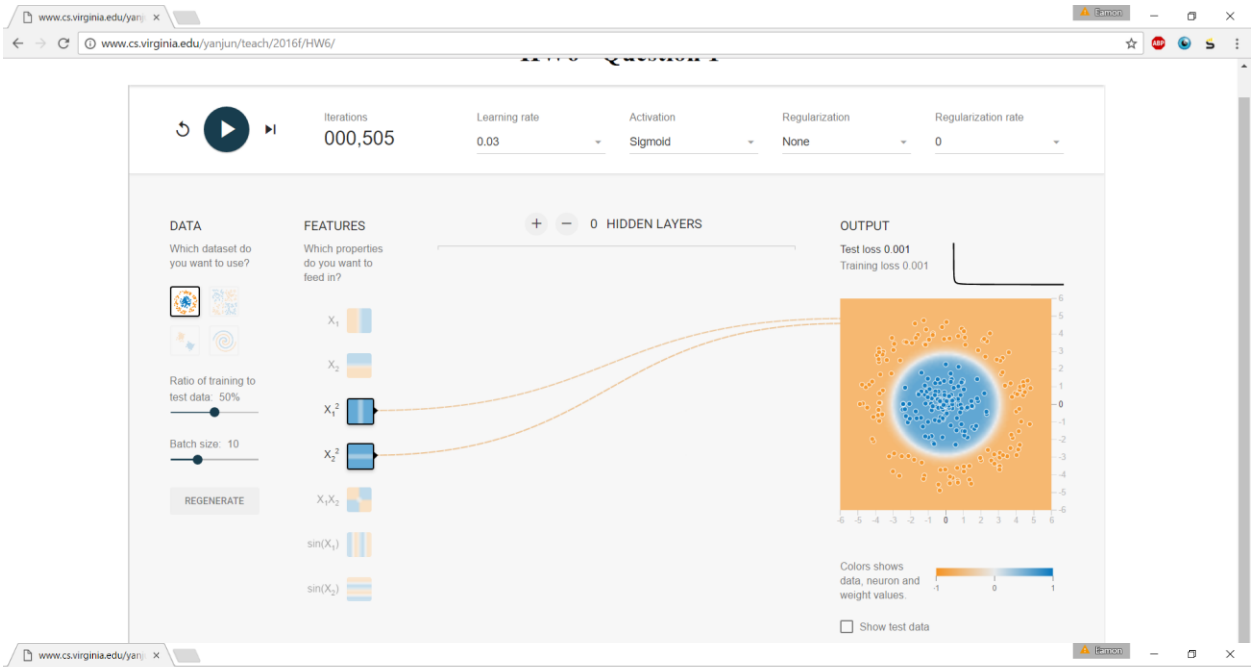
Eamon Collins
Ec3bd
HW6 writeup

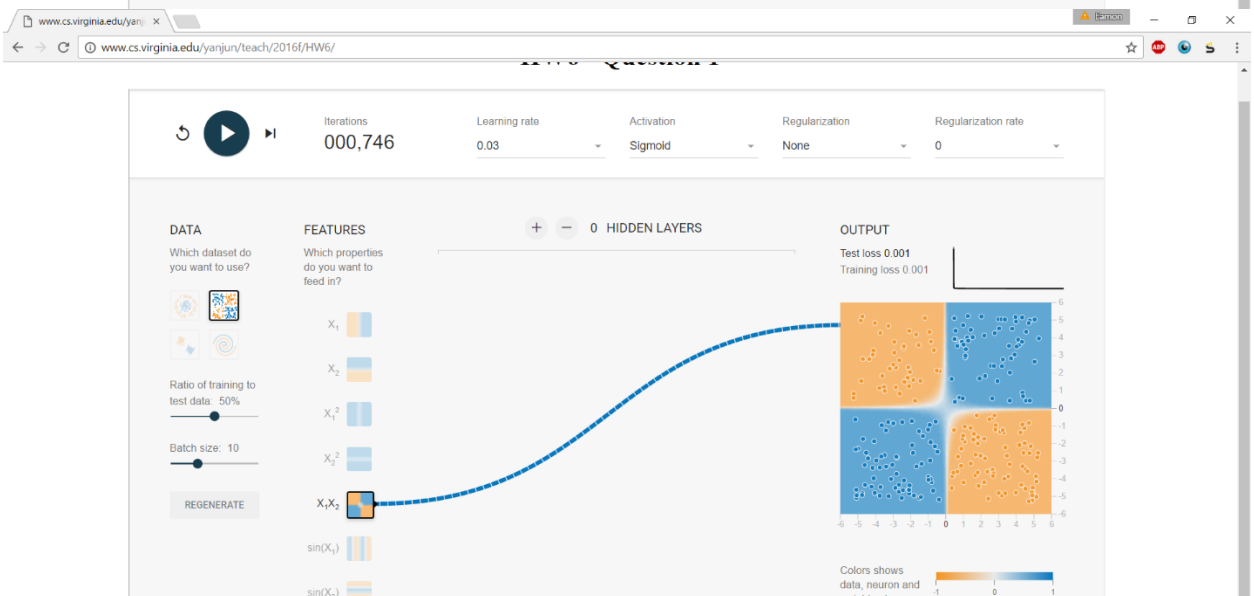
Q1

1.1

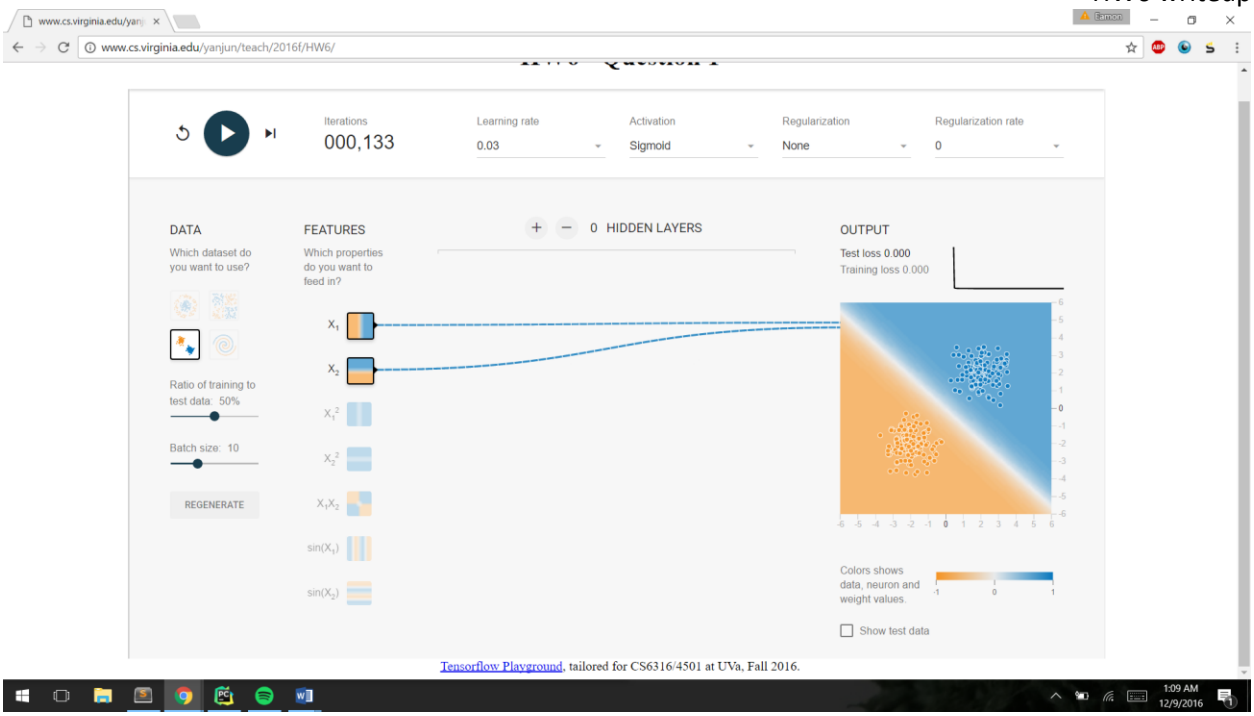
Dataset	Features	Iterations	Test Loss
Circle	X_1^2, X_2^2	505	.001
XOR	$X_1 * X_2$	746	.001
Gaussian	X_1, X_2	133	.000
Spiral	X_2, $\sin(X_1)$, $\sin(X_2)$	602	.479

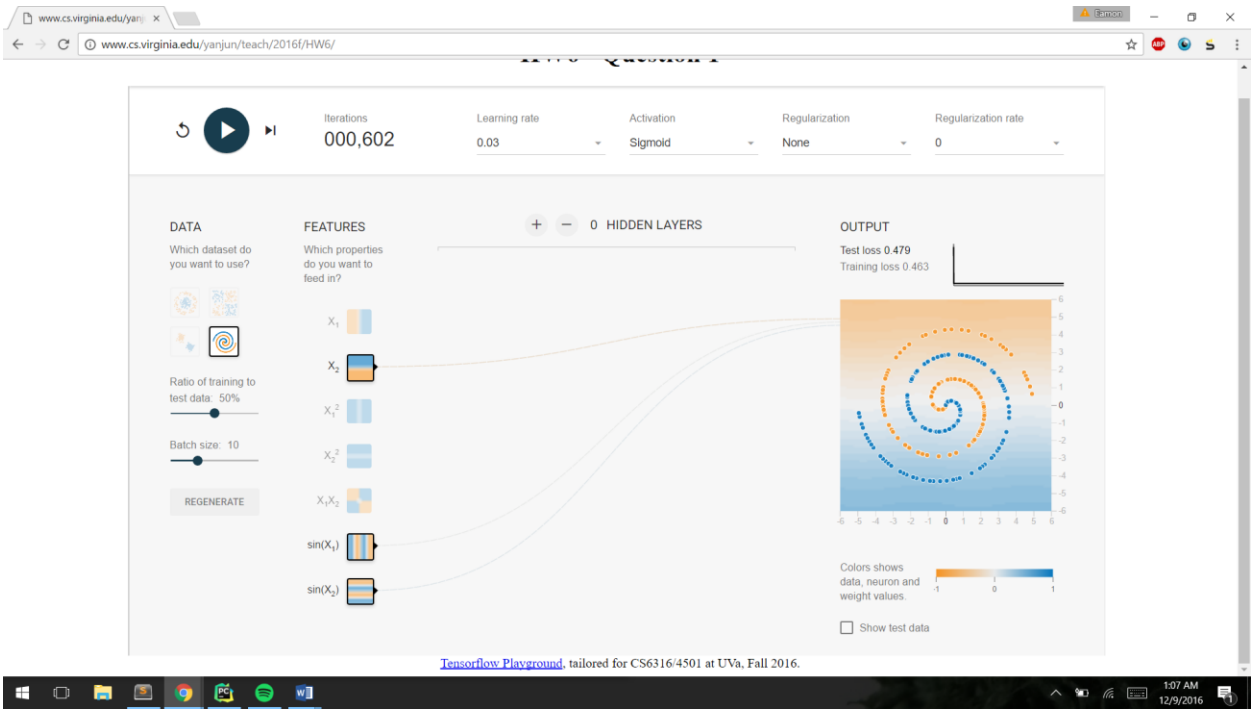
The circle works because x^2+y^2 is the equation for a circle, so the two features squared is all you need to separate them. For the XOR, they are effectively split up by x*y because the sign of the result lines up so neatly with that of XOR. The Gaussian distribution is well separated along a diagonal, so all it needs is x and y to separate it linearly and it can get a very good accuracy result. The spiral one is difficult to predict with the constraints we have been given.





Eamon Collins
Ec3bd
HW6 writeup





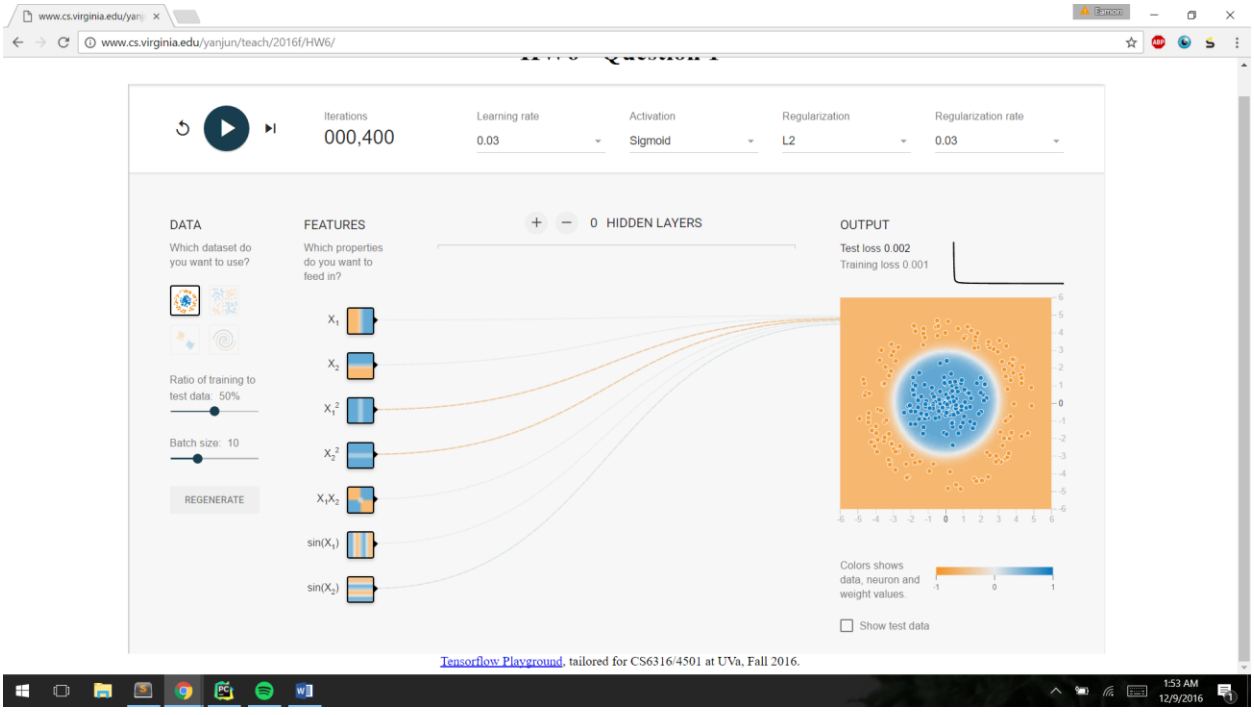
Eamon Collins
Ec3bd
HW6 writeup

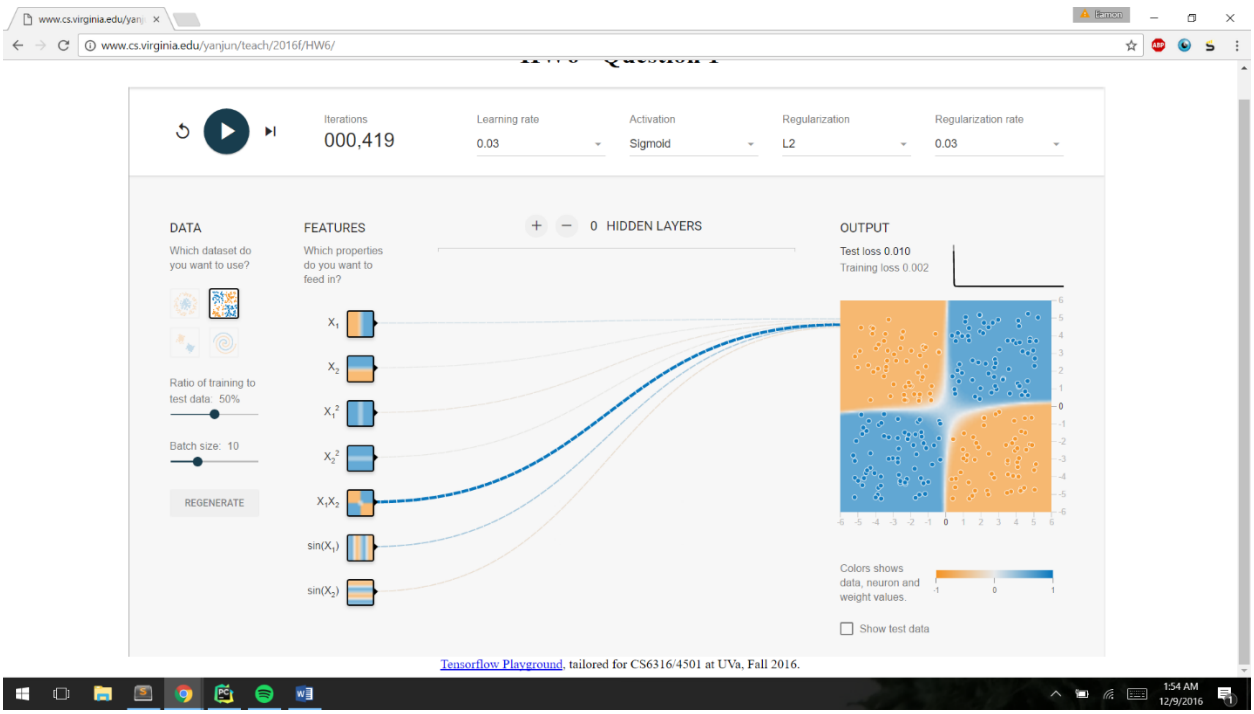
1.2

All numbers are loss scores for all features at reg. rates of .03 and approximately 400 iterations

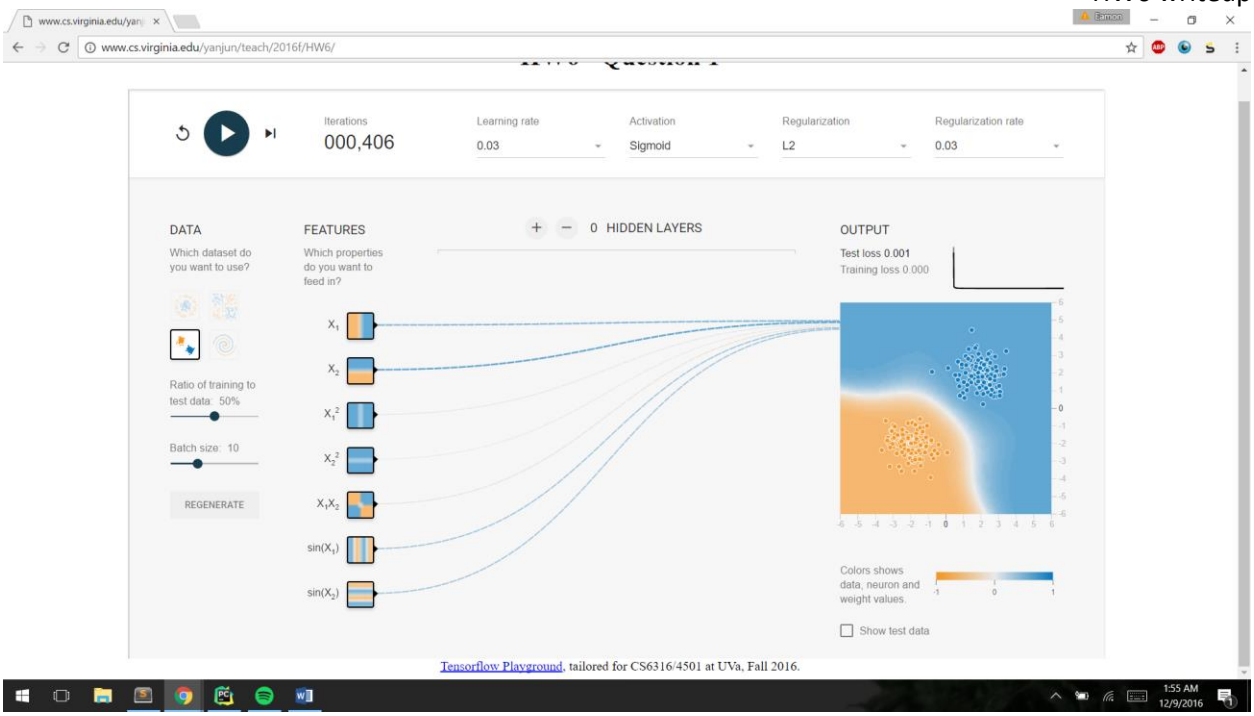
Dataset	No Reg	L1	L2
Circle	.001	.001	.002
XOR	.001	.001	.010
Gaussian	.000	.001	.001

In the experiments, the decision boundary was largely unaffected by the regularization except for the Gaussian. The Gaussian distribution's orange-classification region shrunk dramatically with regularization, I'm not entirely sure why and I don't get what information we're supposed to provide for this question all that well.





Eamon Collins
Ec3bd
HW6 writeup



L1 regularization features selected:

Dataset	Significant features
Circle	X_1^2, X_2^2
XOR	$X_1 * X_2$
Gaussian	X_1, X_2

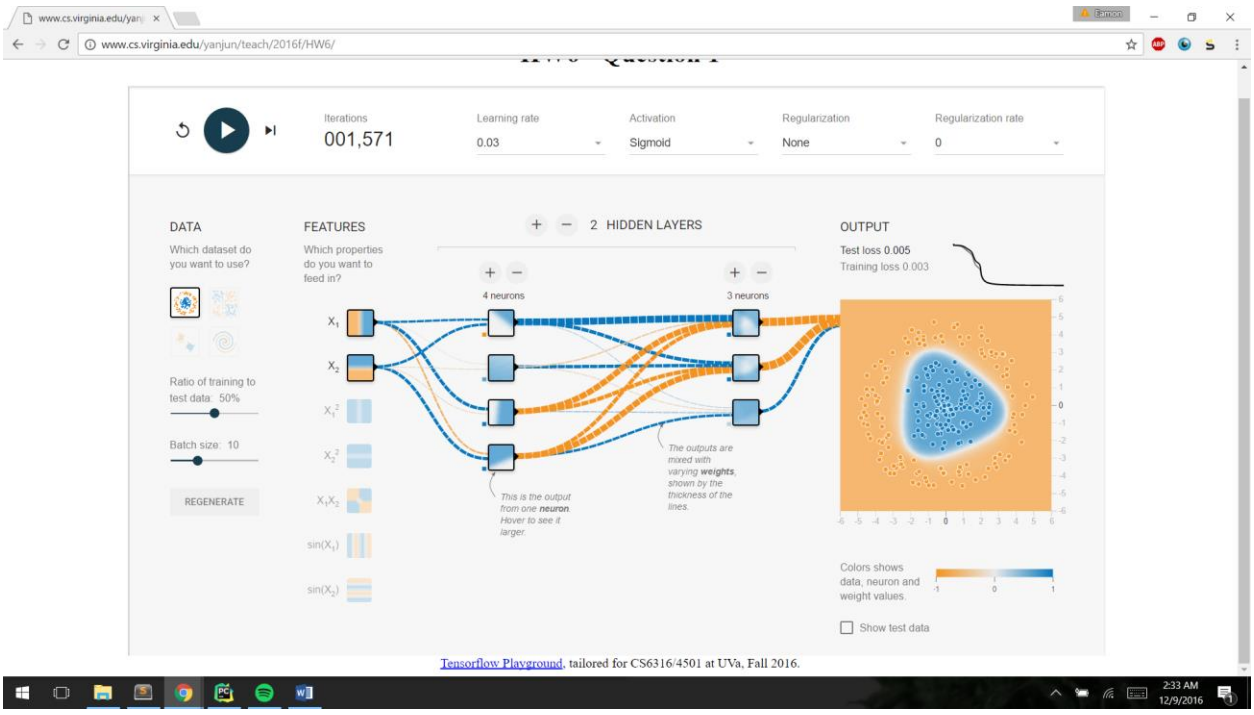
My estimation of these features was exactly correct, partly because they are intuitive and partly because I figured out you could look at the weights for relative importance pretty quickly and that factored into my earlier choices. Although L1 regularization does tend to create sparse representations of the data, zeroing out unimportant features, unregularized neural nets also exhibit the practice of reducing weights on unimportant features and emphasizing the important ones, so it is no surprise the results are similar.

Eamon Collins
Ec3bd
HW6 writeup

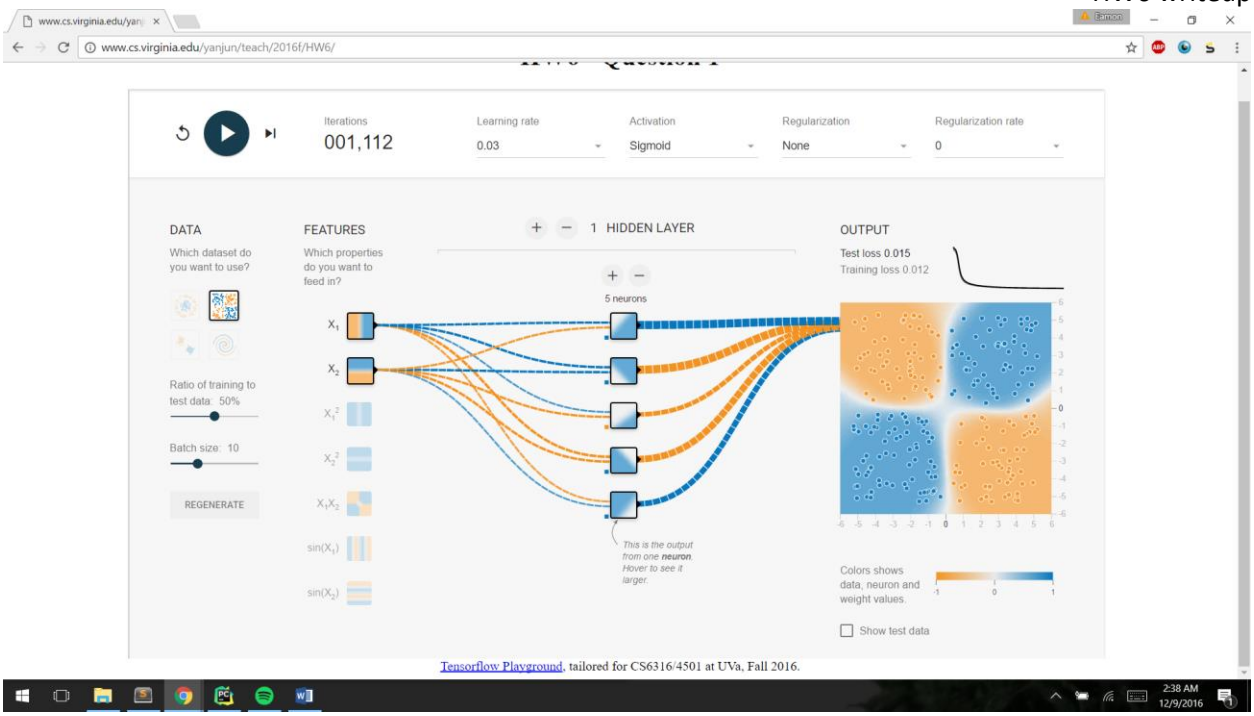
1.3

Dataset	Hidden layers	# neurons at each layer	Iterations	Test loss
Circle	2	4,3	1,571	.005
XOR	1	5	1,112	.015

I had a hard time reaching the same accuracy with more complex models, even with much higher iteration counts. However you can see the neurons each taking on different components of the decision boundary as the weights are refined. For the circle I chose 2 hidden layers because I wanted it to form linear combinations of the original lines to get slanted decision boundaries, and then combinations of those to be able to form curves and it worked somewhat, even if the decision boundary is a little shy of perfectly circular. For the XOR I simply wanted combinations of the simple 1-dimensional decision boundaries, so I only put in 1 layer because I knew it would be sufficient to learn a boundary of that complexity. Then I kept adding neurons until it looked like it was somewhat working.

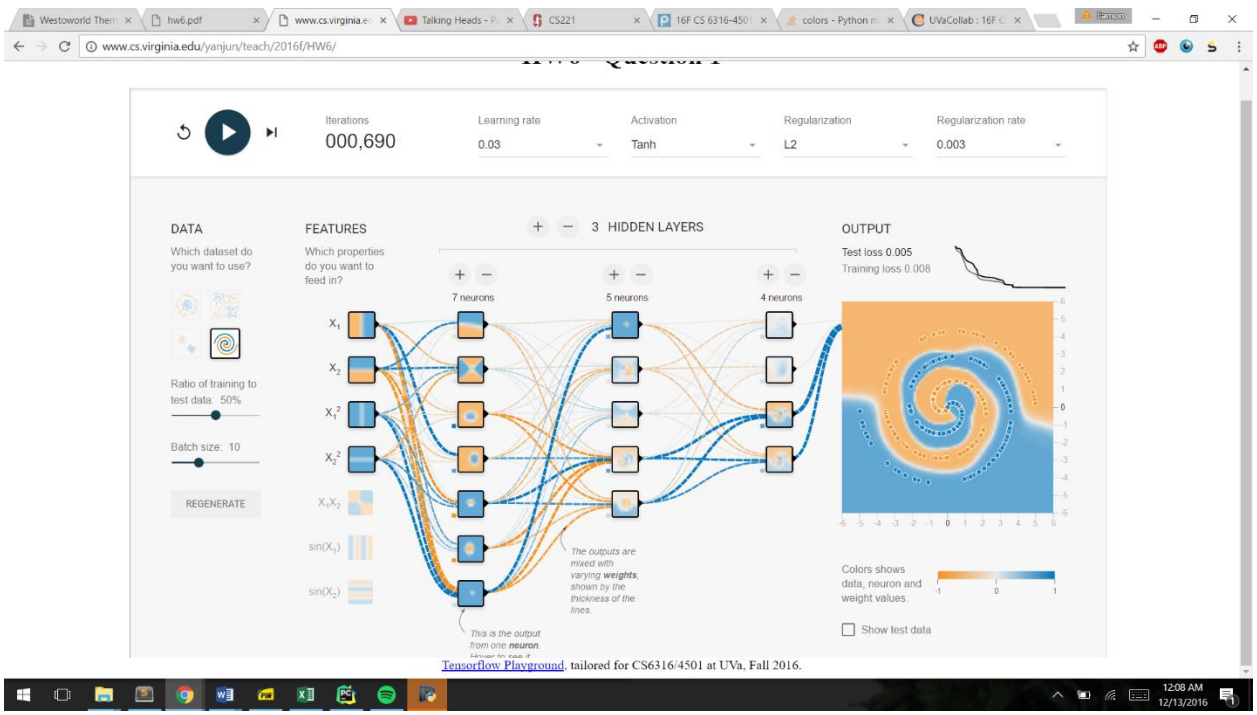


Eamon Collins
Ec3bd
HW6 writeup



Extra credit:

Features	Iterations	Layers	Activation	L2 reg rate:	Test loss
X_1, X_2, X_1^2, X_2^2	690	7-5-4	Tanh	.003	.005



Eamon Collins
Ec3bd
HW6 writeup

Q2

Decision Tree

Parameters as decided by a GridSearchCV:
Criterion = entropy
Max_depth = 15
Min_samples_split = 2
Test accuracy at these values: 0.838565022422

The criterion attribute selects what measure of purity is used when deciding what feature to use to split at each internal node of the decision tree. Entropy means that it is based on the information gain each feature will produce. The max_depth sets the maximum depth of any leaf node. If purity has not been reached before the max_depth, it cuts node production off on that branch and predicts the majority class at each leaf node. The min_samples_split is similar in that it is a pruning method for the tree, it decides the minimum number of samples needed to split any internal node, higher numbers protect against overfitting but the gridsearch chose the default (and minimum) value of 2 for this dataset.

K Nearest Neighbors

K	Train accuracy	Test accuracy
1	1.0	0.943697060289
2	0.983678507749	0.941205779771
3	0.986695926485	0.944693572496
4	0.980661089014	0.943198804185
5	0.979152379646	0.944693572496
7	0.974214785352	0.941704035874
9	0.970923055822	0.937219730942

I would choose a K of 3 because it is tied for the highest accuracy on the test set. It also makes sense that the K is small, because there is relatively low intra-class similarity in this dataset.
This low intra-class similarity is what contributes to such good performance by the KNN technique, as it can classify based on even very small, separated clusters of the same class. Distance weighting, that is, using the whole training set and simply weighting it by how far it is from the point in question, would slow the execution down and not contribute to the accuracy much, if at all. Due to the structure of the dataset I described earlier, it is really only the points close by any given point that are useful.

Eamon Collins
Ec3bd
HW6 writeup

SVM

Kernel	C	Degree	Gamma	Accuracy
Rbf	15		1/p	0.950174389636
Linear	1		1/p	0.926258096662
poly	15	3	1/p	0.953163926258

Theoretically I do think SVM is suited to this problem, as the ability to modulate how you penalize misclassifications can help for the extra noisy samples. Also, we just know that we have more samples than features, and that SVM is a very good general-purpose classifier. For all these reasons I would intuitively agree that a SVM is a good approach.

PCA

KNN

K	# PCs	% variance included	Accuracy
3	6	0.477454341173	0.810164424514
3	17	0.704020654226	0.933233682113
3	25	0.77842588674	0.946188340807
3	40	0.856701859525	0.951170901844

PCA is a good tactic with KNN because in KNN the closer data points of the same class are to each other relative to other classes the better your predictions will be, and when your axes are built of the features of highest variance, it will theoretically separate different classes from each other more. This is supported because at high PC counts we not only achieve the lower feature count inherent to the PCA technique, we actually are able to classify with better accuracy across our test set than the non-PCA KNN accuracy of 0.944.

SVM

Svm parameters used: kernel=poly, C=15, Degree=3, gamma = 1/p

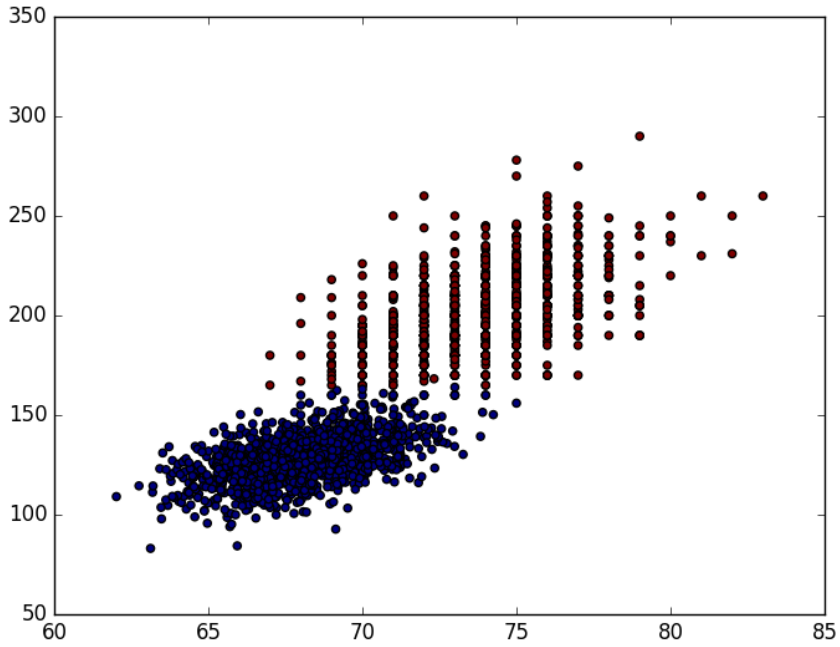
# PCs	% variance included	Accuracy
6	0.477454343704	0.801694070752
17	0.704020049582	0.934728450424
40	0.856701859525	0.954160438465

PCA increases my test accuracy for SVM, but only by a little bit. I would have expected it to do very well for similar reasons as the KNN. The further you separate the classes (by using high variance axes) the easier it will be to find a wide margin, which is what the SVM is essentially trying to do. Essentially I think PCA is overall a good tactic with this dataset as it can reduce the effect of the (smaller) variances between handwriting and pronounce the larger variances that occur between different numbers.

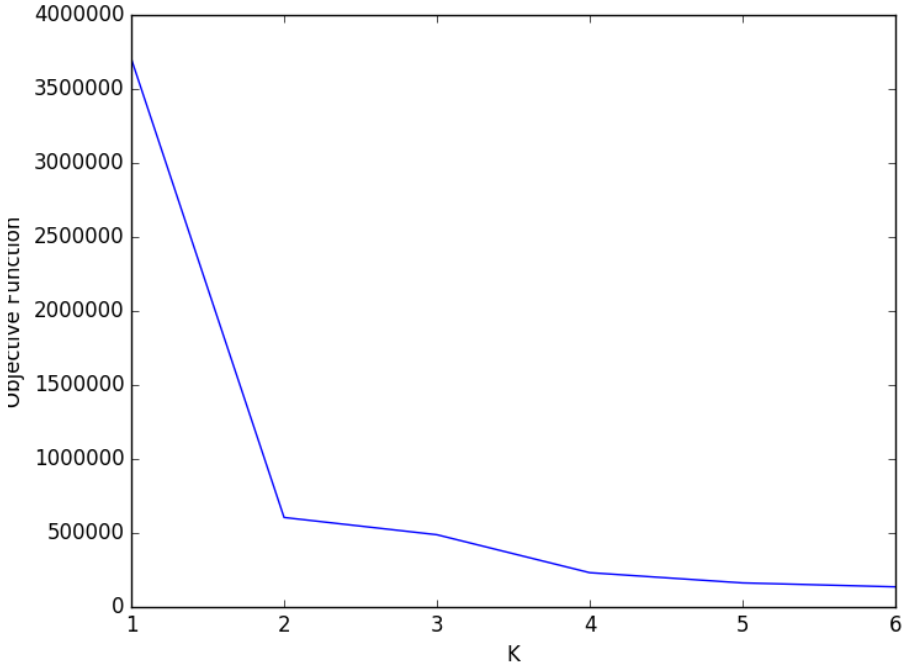
Eamon Collins
Ec3bd
HW6 writeup

Q3

Q3:



Q4:



Eamon Collins
Ec3bd
HW6 writeup

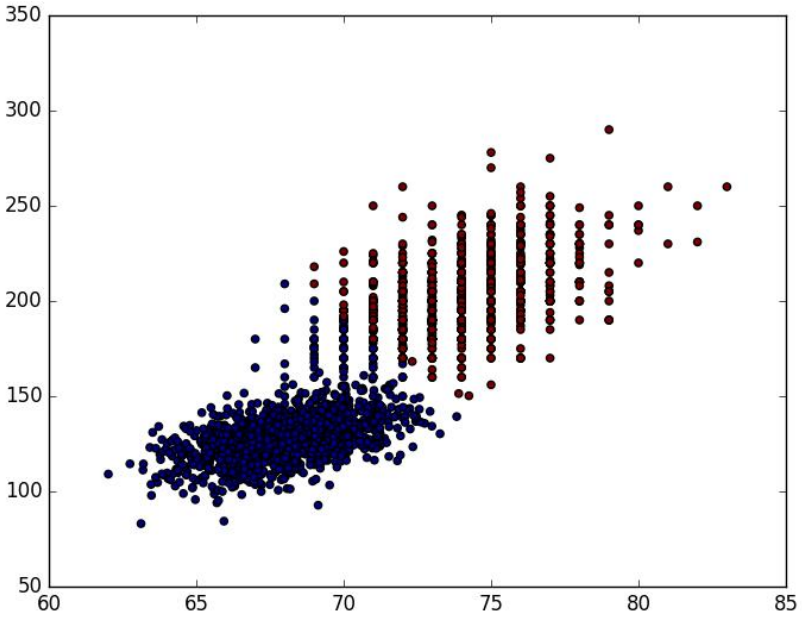
The graph clearly suggests K=2 is the best pick of K for this data.

Q5: using the K=2 value found in our knee-finding process, these are the purity values of those 2 clusters after 50 iterations:

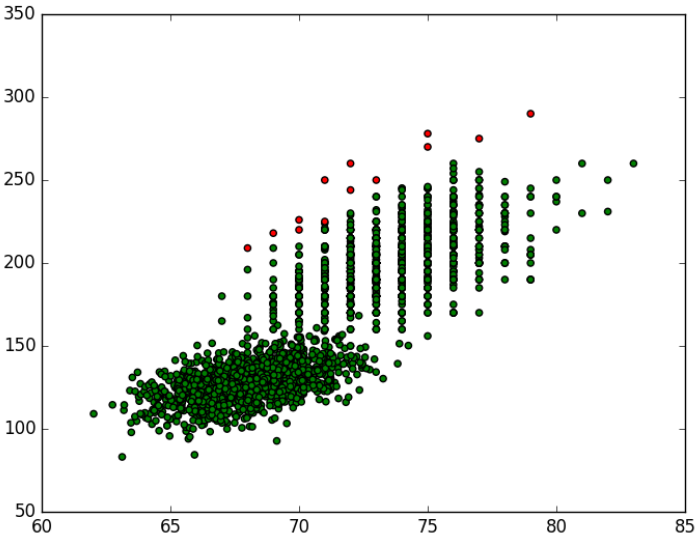
Purity values for each cluster: [0.97957516339869277, 0.99900990099009901]

Q7:

Diag, color differentiation here is red and blue:



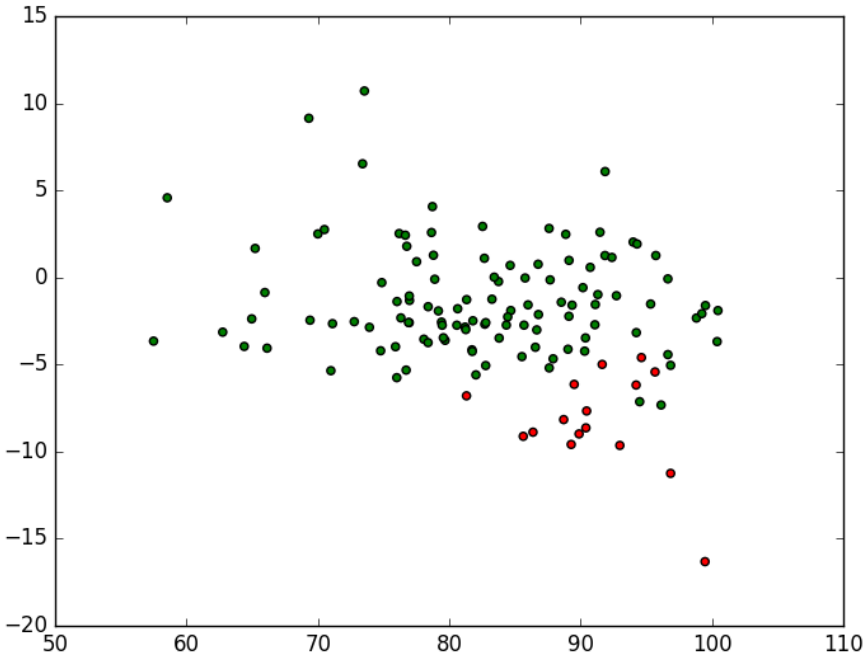
Full:



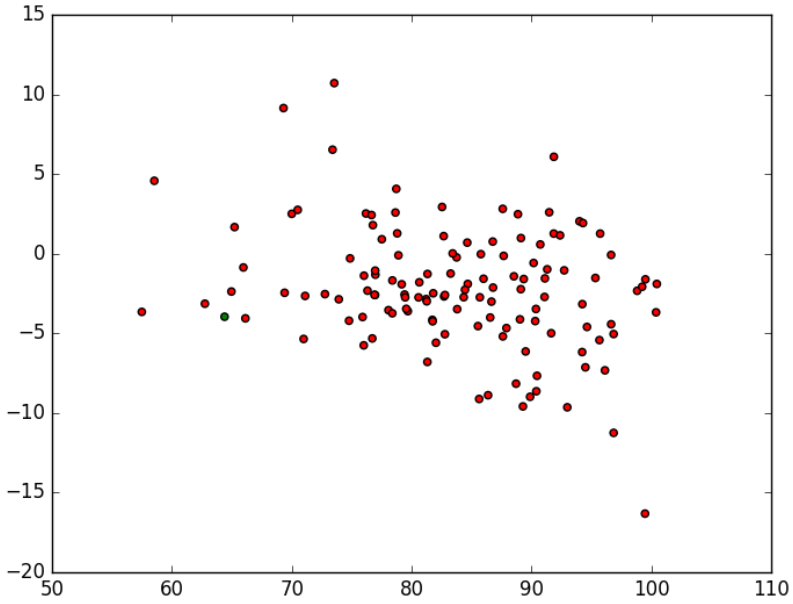
Eamon Collins
Ec3bd
HW6 writeup

Q8:

Diag:



Full:



Eamon Collins
Ec3bd
HW6 writeup

Q9:

Purities:

Dataset	Covariance type	Iterations	Purity Group 1	Purity Group 2
Dataset 1	Diag	20	.93	.9779
Dataset 1	Full	20	1.0	0.54029716343989198
Dataset 2	Diag	200	0.6875	0.5267857142857143
Dataset 2	Full	200	1.0	0.50393700787401574

I think I may have a problem with my full covariance matrix, to do with how they were formed interacting with how I was storing them as an np.matrix. However, they do classify things, just not very well. If there is no problem there, the iteration count may just be low