6.867: Homework 1

1 Gradient Descent

We started by implementing gradient descent, a numerical technique that is commonly used to optimize differentiable functions. Although gradient descent is typically vulnerable to getting trapped in local minima, the two objective functions chosen for our experiment are quasiconvex and as such, the algorithm should always converge on the global minimum regardless of initialization.

To better understand how the starting guess, step size, and convergence criterion hyperparameters affect the performance of the gradient descent algorithm, we conducted several experiments where we adjusted each of these parameters, keeping track of both the value of the objective function over time and the time required for the algorithm to satisfy the convergence criteria.

1.1 Starting guess

The optimal values, according to the closed form solution, for the given negative Gaussian and quadratic bowl are (10, 10) and (400/15, 400/15) respectively. Since both functions are symmetric and the behavior is identical on both axis, we chose a series of starting guesses whose Euclidean distance from the optimal value increased linearly along the first axis.

1.1.1 Negative Gaussian

As shown in Figure 1, the value of the Gaussian objective function decreases monotonically and approaches the same minimum regardless of the starting guess. However, the time required to satisfy the convergence criteria - when the norm of the gradient is less than 1e-3 - increases along with the distance between the starting guess and the optimal values.

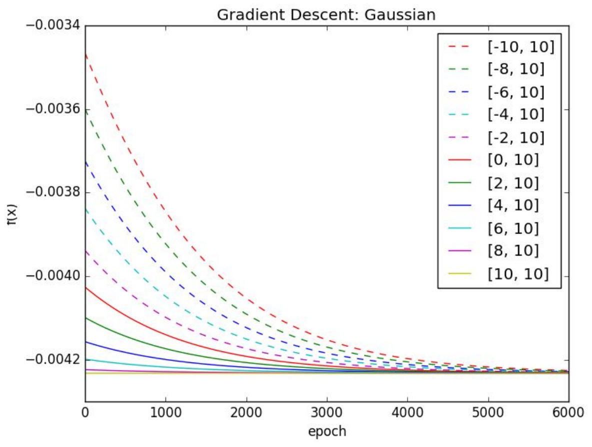

Figure 1. The value of the objective function (negative Gaussian) vs time for a set of starting guesses.

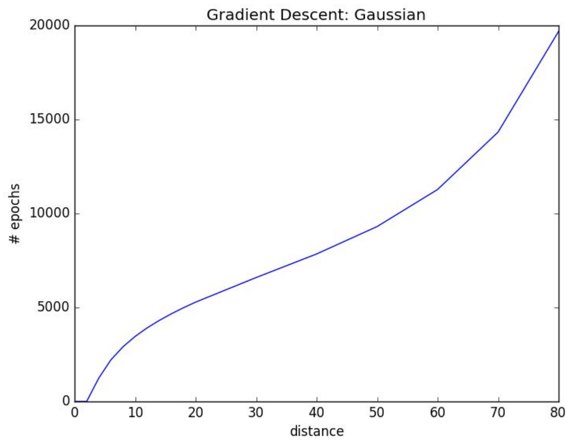

Figure 2. The number of cycles to converge vs the distance between the guess and the optimal value.

The training time appears to increase logarithmically at first but then accelerates - this can be explained by the fact that the gradient of the Gaussian approaches zero as the distance from the mean increases, making our adjustments smaller. This problem can be solved by using adaptive learning rates such as adagrad and rmsprop which incorporate ideas such as momentum to adjust the size of the gradient.

By looking at how the norm of the gradient changes over time, we can gain a better understanding of how the gradient affects the training time for simple gradient descent. Figure 3 shows how the norm of the gradient behaves for various starting points. When the starting guess is far away, the norm increases initially as we move from the near-zero regions of the Gaussian into the active region; this accounts for the drastically increased training times when the starting guess is especially bad.

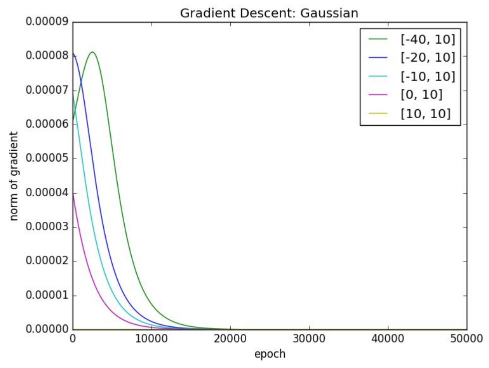

Figure 3. The norm of the gradient over time for a set of starting guesses.

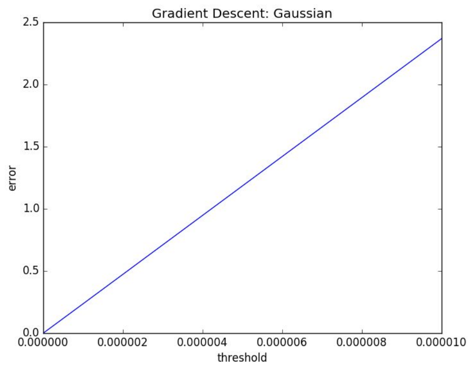

Figure 4. The distance between the learned result and the optimal value vs the threshold.

The relationship between the convergence threshold and the mean squared error of the final estimate is shown in Figure 4, where the two variables appear to be directly proportional. This linear relationship helps us determine thresholds which will provide a balance between the convergence time and the accuracy of the final result.

1.1.2 Quadratic Bowl

Unlike the negative Gaussian, the gradient for the quadratic bowl does not approach zero as the distance between the starting guess and the optimal value increases. This is reflected in the figure on the right where the number of cycles required to converge appears to increase logarithmically.

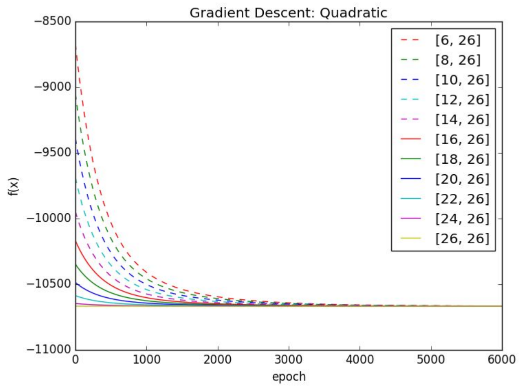

Figure 5. The value of the objective function (quadratic bowl) vs time for a set of starting guesses.

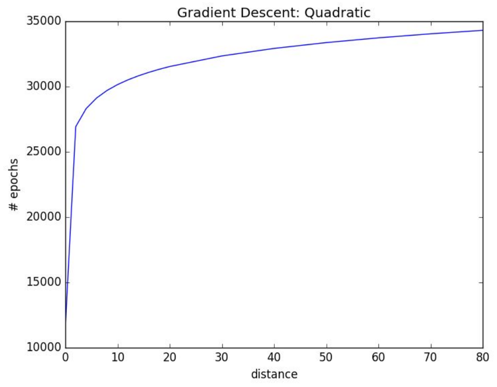

Figure 6. The number of cycles to converge vs the distance between the guess and the optimal value.

The norm of the gradient decreases monotonically for the quadratic bowl - this explains why gradient descent converges faster on this objective function than it did on the negative Gaussian.

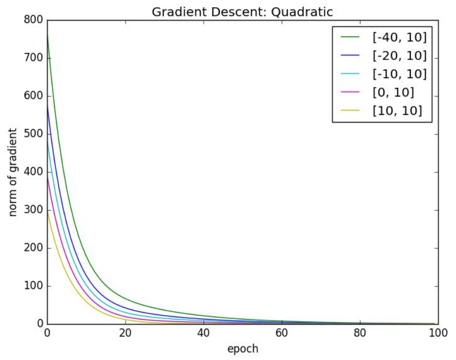

Figure 7. The norm of the gradient over time for a set of starting guesses.

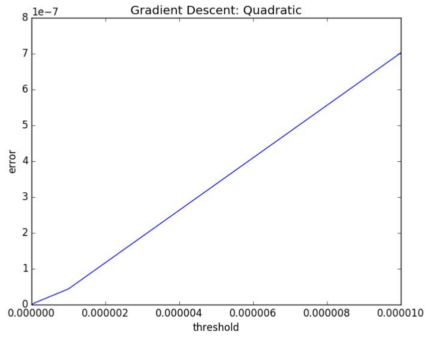

Figure 8. The distance between the learned result and the optimal value vs the threshold.

1.2 Step size

For both objective functions, a larger step size generally resulted in faster convergence. A notable exception is that when the step size is too large, gradient descent fails to converge on the quadratic function and instead oscillates around the optimal value before diverging and overflowing.

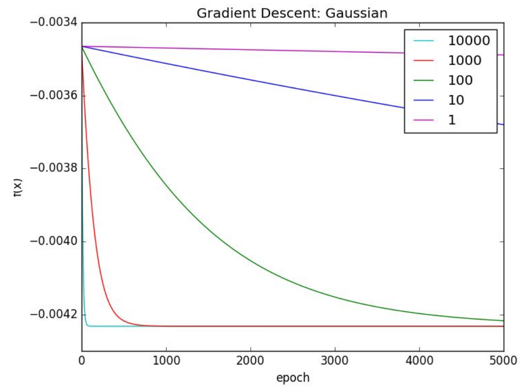

Figure 9. The value of the Gaussian objective function over time for various step sizes.

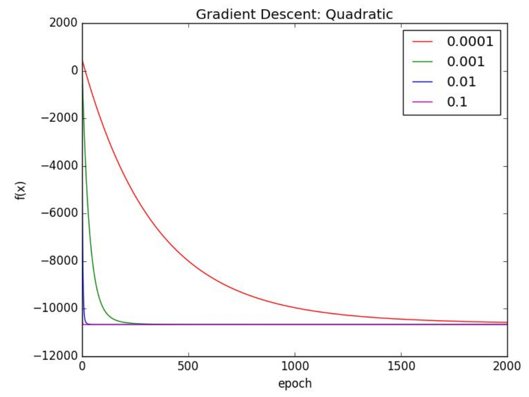

Figure 10. The value of the quadratic objective function over time for various step sizes.

1.3 Derivative approximation

Regardless of the objective function and barring numerical problems due to underflow, a smaller δ value always results in a better approximation of the gradient. However, when we reran the above experiments, we determined that even very rough estimates were sufficient for the gradient descent procedure to converge.

δ	X	Gaussian actual	Gaussian estimate	Quadratic	Quadratic estimate
100	[16,5]	2.463e-5, -2.052e-5	1.761e-7, -1.441e-7	-215, -270	-215, -270
10	[16,5]	2.463e-5, -2.052e-5	2.344e-5, -1.953e-5	-215, -270	-215, -270
1	[16,5]	2.463e-5, -2.052e-5	2.462e-5, -2.051e-5	-215, -270	-215, -270
.1	[16,5]	2.463e-5, -2.052e-5	2.463e-5, -2.052e-5	-215, -270	-215, -270

Figure 11. The actual and estimated values for both the Gaussian and quadratic functions for various δ values.

1.4 Batch, Stochastic, Mini-batches

To evaluate the performance of batch, stochastic, and mini-batch gradient descent, we modified our gradient descent algorithm to support arbitrary batches sizes and then set the size to 100, 50, 20, and 1. Since the dataset contains 100 entries, using mini-batches of size 100 is equivalent to performing full batch gradient descent, and using mini-batches of size 1 is equivalent to pure stochastic gradient descent.

To ensure convergence, we chose a learning rate schedule determined by (100000 + epoch) ** -0.5, where an epoch indicates a single iteration over the entire training data set. For example, an epoch for batch size 100 would involve computing 1 gradient while an epoch for batch size 1 would require computing 100 gradients.

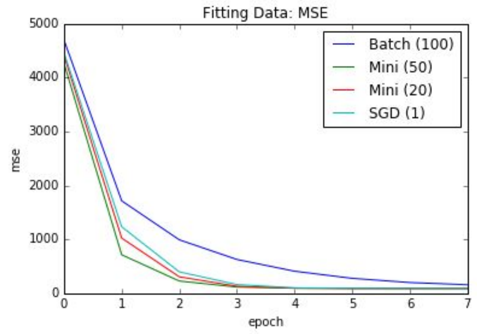

Figure 12. MSE over time for mini-batch sizes.

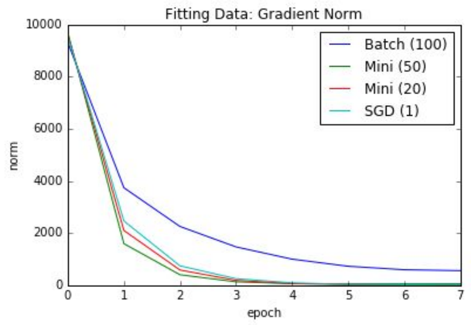

Figure 13. Gradient size over time for mini-batch sizes.

Based on the above plots, we conclude that stochastic and mini-batch gradient descent significantly outperform the batch gradient descent algorithm, converging in fewer epochs. It appears that even though stochastic gradient descent - unlike batch gradient descent - does not use the true gradient, the pointwise gradient for each of the data points is sufficient for it to converge on a final weight vector that is a reasonable approximation of the maximum likelihood weight vector.

2 Linear Basis Function Regression

We started by applying batch and stochastic gradient descent to the data set, transforming the features using a Mth-order polynomial basis. For low M values, both techniques converged in a similar timeframe and eventually reached the optimal value as determined by the closed form solution; in addition, arbitrarily chosen initial guesses all resulted in convergence a reasonable amount of time.

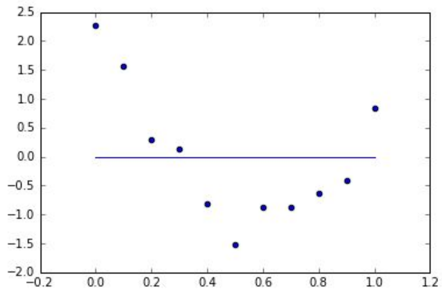
Figure 14. M = 0

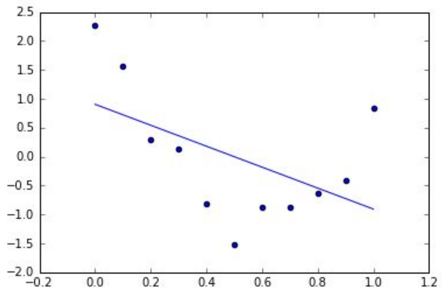
Figure 15. M = 1

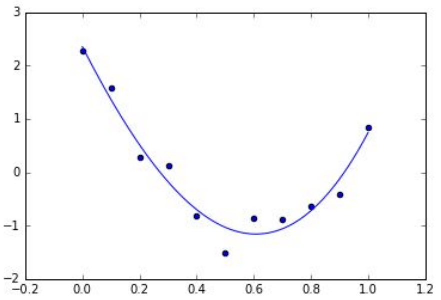
Figure 16. M = 2

The step size, on the other hand, had a significant impact on the time required to converge. Based on experiments, it appears a relatively large step size between 0.5 and 1.0 results in the fastest convergence time for batch gradient descent. We also experimented with using stochastic gradient descent with a dynamic step size specified by (1 + t) ** -0.5 but were unable to achieve significantly faster convergence than the baseline set by batch gradient descent.

After setting M to a large value such as 10, we encountered some difficulty getting gradient descent to converge in a reasonable amount of time without implementing more advanced adaptive learning rates. The objective function is quasiconvex so gradient descent is guaranteed to converge, with some restrictions on the step size, but because the optimal weight vector for the 10th order polynomial contains values in the order of 10^5, it is unlikely to converge in a reasonable amount of time without a finely-tuned learning rate schedule.

The figure on the right shows the result applying linear regression with a cosine basis on the data set. Even without regularization, our model provides a reasonable estimate of the weight vector.

Actual Weights
[1. , 1.5, 0. , 0. , 0. , 0. , 0. , 0.]

Estimated Weights
[0.77, 1.09, 0.10, 0.14, -0.05, 0.36, 0.01, 0.02]

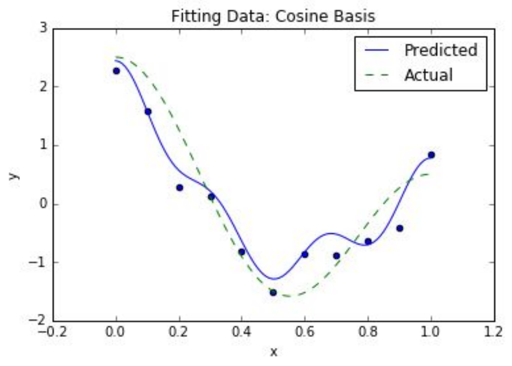

Figure 17. Fitting the data with a cosine basis.

3 Ridge Regression

We implemented the closed form solution for ridge regression (L2 regularization) and fit Mth order polynomials to the previous dataset. In general, larger λ values resulted in a flatter function that was less likely to overfit to the training data, increasing bias while decreasing variance. Without a validation data set, we were unable to determine the ideal λ and M values but a visual analysis shows that carefully chosen λ values can result in a better fits, where better is defined as being more likely to generalize well, than unregularized (λ = 0) regression.

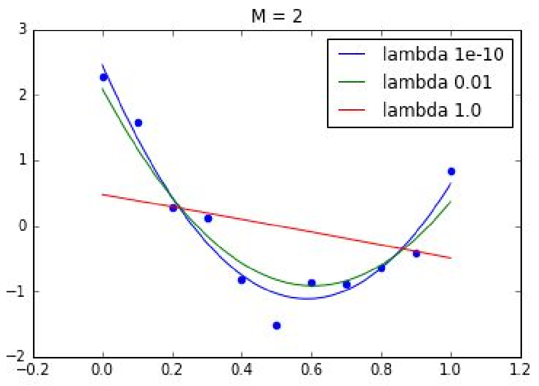

Figure 18. M = 2, various λ

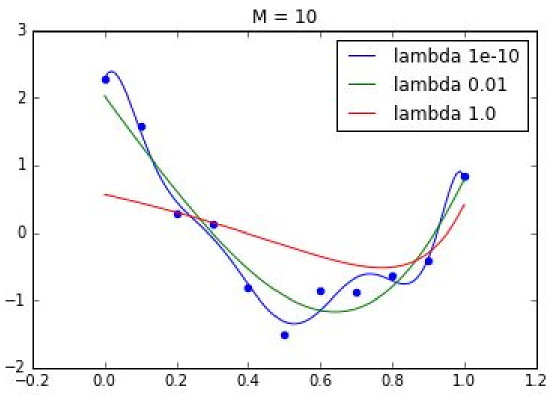

Figure #. M = 19, various λ

In order to better understand how M and λ affect the resulting function's ability to predict new values, we trained our model on the ridge regression dataset and ran a grid search over the two hyperparameters, seeking values that would minimize the validation loss.

M	λ	Train A MSE	Validation MSE	Testing B MSE
1	.00001	0.99	1.70	5.01
1	1.0	1.01	1.80	5.03
2	**.00001**	**0.93**	**1.53**	**5.07**
2	1.0	0.95	1.61	5.09
3	.00001	0.93	1.54	5.07
3	1.0	0.99	1.85	5.15
12	.00001	0.03	7.29	6.27
12	1.0	0.85	2.72	4.74

Figure 20. M and λ values using A as training data.

M	λ	Training B MSE	Validation MSE	Testing A MSE
1	.00001	3.79	5.93	4.86
1	1.0	3.82	5.70	4.62
2	.00001	3.49	6.21	5.63
2	1.0	3.52	5.83	5.22
3	.00001	3.13	5.50	6.99
3	**1.0**	**3.26**	**5.21**	**5.81**
12	.00001	0.15	14881	11648
12	1.0	0.65	80.4	78.3

Figure 21. M and λ values using B as training data.

As shown on the figure on the right, increasing λ results in a larger training error but reduces the validation error, increasing bias by decreasing variance. The result is a model which generalizes well. After a certain point, however, increasing λ no longer produces a noticeable difference in the validation error.

By choosing λ such that we minimize the validation error, we can produce a model which will work well on new, previously unseen data points.

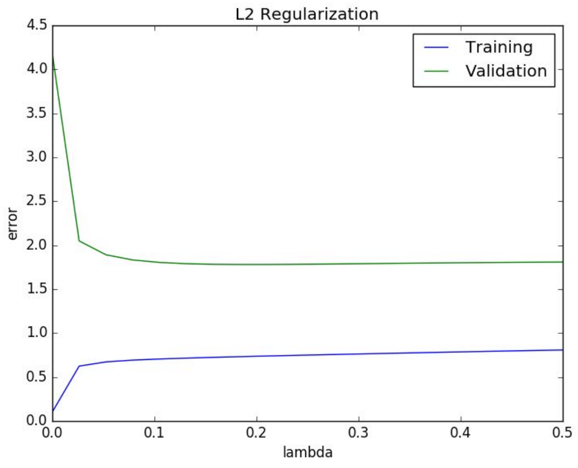

Figure 22. MSE vs λ for M=10

4 LASSO Regression

Using the L1 regularizer as implemented by the sklearn.linear_model package, we obtained a sparse weight vector where all but four of the weights were zero. These non-zero coefficients corresponded with the non-zero coefficients in the true basis. To get a better idea of how this compares to both unregularized and ridge regression, we applied all three techniques to this data set and observed that LASSO appeared to provide the best fit, resulting in a model that was most similar to the true function.

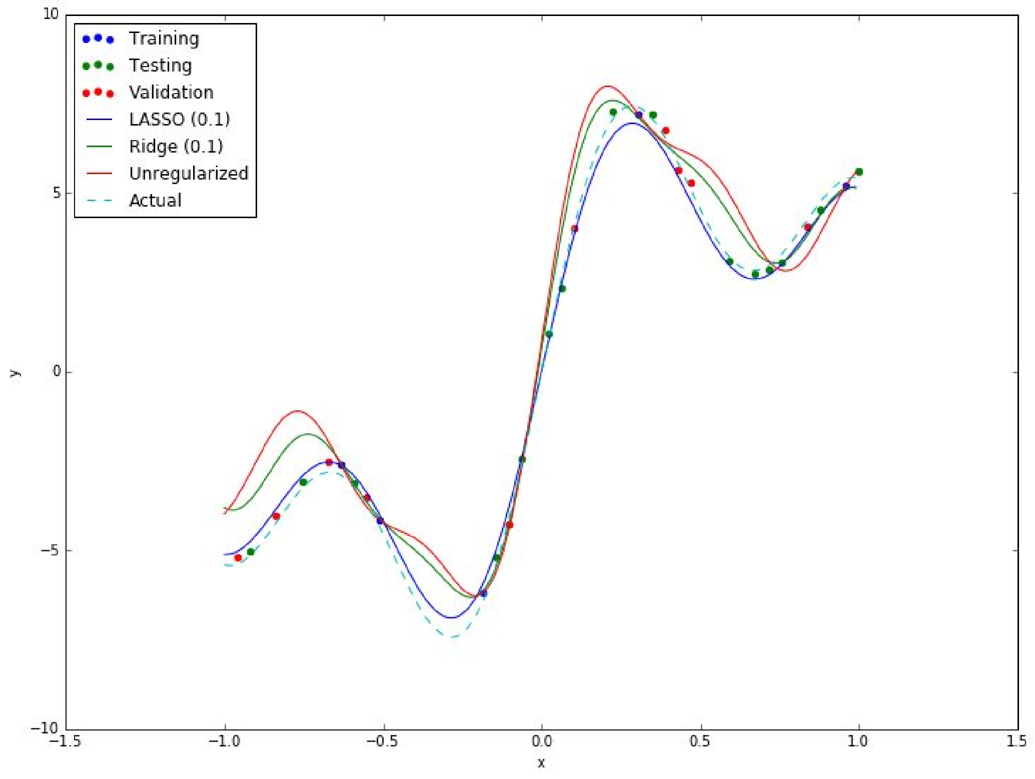

Figure 23. This figure shows the four functions and the three datasets.

To compare how each of the regularization techniques affects the final weight values, we plotted the 13 coefficients for the true, LASSO, ridge, and unregularized functions. The chart below shows that while both LASSO and ridge regression result in better approximations of the true weights than the unregularized regression, only LASSO regularization correctly discards the irrelevant dimensions of the sinusoidal basis.

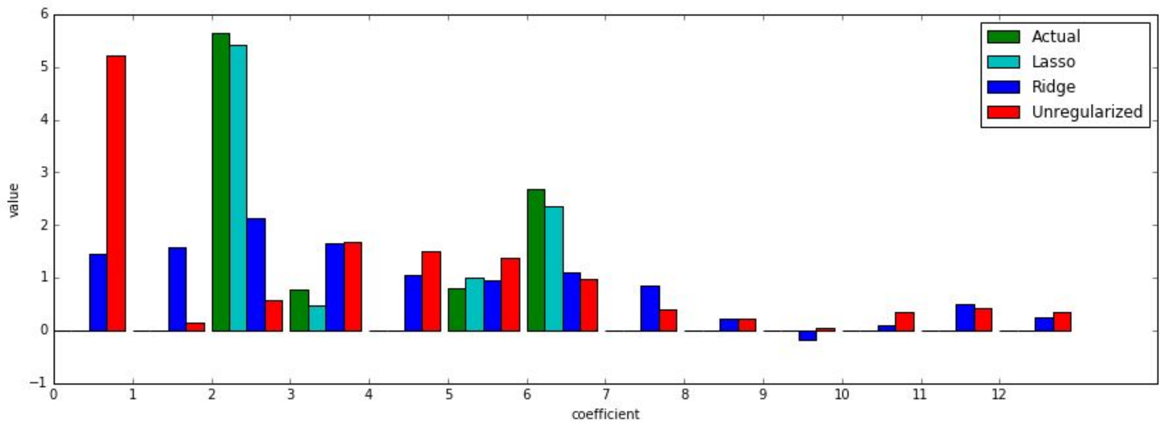

Figure 24. This figure shows the final weights that each of the models converges on.