6.867 Pset 1

1 Implement Gradient Descent

1.1 Gradient Descent

Our group implemented a very basic gradient descent procedure. The algorithm is a function of five arguments, the objective function $f(x)$, gradient function $g(x)$, initial guess g, a step size s, and termination threshold t. The algorithm operates with the following rules:

1. Initialize $x_0 = g$
2. Update $x_{i+1} = x_i - sg(x_i)$
3. Terminate when $|f(x_i) - f(x_{i-1})| < t$.

To verify that our procedure exhibited the desired behavior, we tested it on the following functions.

$y = x^2$ As expected, we found that all initial guesses converged to $x = 0$ with precision varying inversely with t. $t = 0.001$ yielded $x = 0.156$ while $t = 10^{-6}$ yielded $x = 0.005$. With reasonable step sizes, the yielded x matched sign with the initial guess.

$y = x_0x_1$ When run with $g = [0.0001, 0.0001], t = 0.001$ we found x converged to $[0, 0]$ as expected. Similarly, with $g = [50, 50], t = 0.001$ we found that the algorithm converged and terminated at $x = [0.22, 0.22]$. However by changing the initialization to $g = [100, 50]$, the optimization "over-shoots" and since there is no global minimum x approaches $[-\inf, \inf]$.

$y = (x_0 - 5)^2 + (x_1 + 3)^2$ Here we verified that our method finds multi-valued non-zero minima. All converging input values correctly approached $x = [5, -3]$

$y = (x^2 + 3x + 2)^2$ We were able to verify that our gradient descent was able to converge to multiple local minima. For values g greater than -1, x converged to -1 and for values of g less than -2, x converged to -2. When run with $g = -100, s = 0.01t = 10^{-5}$ the step at each point is too high and the the algorithm is unstable, and does not converge.

1.2 Numerical Gradient

To allow us to use our gradient descent procedure on functions which do not have easily computed analytic gradients, we wrote a procedure to approximate the gradient of a function numerically. The numerical gradient function takes as arguments, a function f a value x and a step size s. We computed the partial derivative of f along a dimension i, at point x as

$$\frac{df}{di} = \frac{1}{s}\left[f\big(x + \frac{s}{2}\hat{\mathbf{i}}\big) - f\big(x - \frac{s}{2}\hat{\mathbf{i}}\big)\right]$$

where $\hat{\mathbf{i}}$ is the unit vector along the dimension i. This represents a central difference calculation at x normalized by the step size s. To verify that this function provided the desired behavior, we tested against the above listed functions. This gradient estimator was tested on the following functions and values:

Function	Value	Gradient	Estimate
$y = x^2$	5	10.0	10.0
$y = x_0x_1$	$[-100, 50]$	$[50, -100]$	$[50 + 10^{-8}, -100 + 2 * 10^{-7}]$
$y = x_0x_1$	$[0, 0]$	$[0, 0]$	$[0, 0]$
$y = (x_0 - 5)^2 + (x_1 + 3)^2$	$[3, 3]$	$[-412]$	$[-4.12.]$
$y = (x^2 + 3x + 2)^2$	-9	-1680	-1680.00075

We found near perfect accuracy on all of the tested function. To benchmark the performance of our gradient descent implementation we tested against the scipy minimizer, `fmin_bfgs` on a simple quadratic bowl as seen below.

Guess	Step Size	Threshold	Value (us)	Evals (us)	Value (scipy)	Evals (scipy)
5	0.01	0.00001	0.015	286	-7.4513e-9	12
100	0.01	0.00001	0.016	434	-7.4515e-9	12
100	0.01	0.01	0.483	264	-7.4515e-9	12
100	0.1	0.00001	0.003	46	-7.4515e-9	12

The scipy minimizer out-performed our minimizer in all dimensions; which is to be expected. By implementing more sophisticated techniques we could improve the speed of our minimizer but for the purposes of the remainder of this problem set, correctness is sufficient.

2 Linear Basis Function Regression

2.1 Linear Basis MLE

Our maximum likelihood weight vector computation followed the standard form. Given a set of observation points $X = x_1, x_2, ...x_n$ with observed values $Y = y_1, y_2, ...y_n$, and a desired function order o. We compute a n by $(o+1)$ matrix Φ such that

$$\Phi_{rc} = x_r^{c-1}$$

This leaves us with the problem of finding the weights, W to minimize $\|\Phi W - Y\|_2$. This can be computed using a standard least squares computation. $W = (\Phi^T\Phi)^{-1}\Phi^T Y$. Using this method we computed polynomials of varying order to fit the given data (see figure 1).

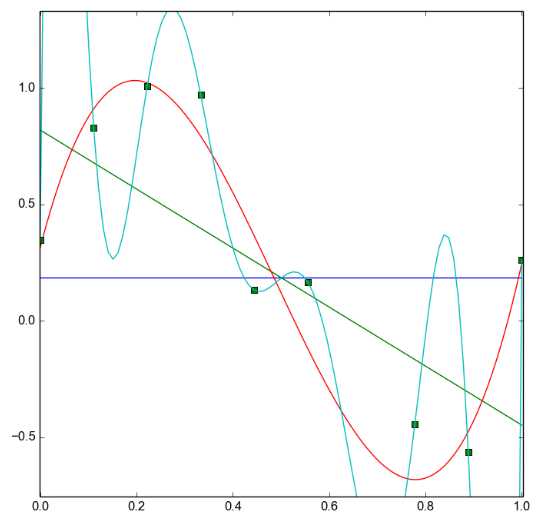

Figure 1: Plot of weighted polynomials, computed with closed from OLS
Blue: Order 0, Green: Order 1, Red: Order 3, Cyan: Order 9

2.2 SSE

We computed the SSE given an input vector X, weights W and output Y as $\|\Phi W - Y\|_2$, where Φ is computed from X as seen above. The gradient of the SSE was computed in the following way. We can write the error as

$$\epsilon = \sum_{i=1}^{n} (y_i - \Phi_i W)^2$$

where Φ_i denotes the i^{th} row of Φ. Thus for a particular dimension, d,

$$\epsilon_d = \sum_{i=1}^{n} 2(y_i - \Phi_i W)(-\Phi_{id})$$

This can be computed for all d giving the final gradient.

To verify this gradient we computed the distance between the gradients produced analytically and numerically on a series of randomly generated orders and weights. Which is to say for a randomly generated vector r, (vales in range $[-100, 100]$) of random order $o \in [1, 9]$, we computed $\epsilon = \|g(r) - g'(r)\|_2$ where g is the analytical gradient function and g' is the numerical gradient function. We found the average value for ϵ to be less than 10^{-8}, which is sufficiently accurate for our purposes.

2.3 Linear Basis Gradient Descent

As we decreased the threshold we found more accurate plots, but the higher order polynomials took progressively longer to compute. With $t < 10^{-7}$ the computation took prohibitively long to compute. With higher step vales, the descent became unstable, while low values were slow. Our final weights seemed not to depend strongly on the initial guesses in general. When we specified guesses which were not artificially close to the optimal values we found roughly the same convergence. We discovered that the parameters we found which most accurately recreated the bishop plots where $t = 10^{-7}, s = 0.01$ and initial guesses with weight 1 for all dimensions (see figure 2). When we computed the optimal weights using the built-in `fmin_bfgs`, we found that it converged to roughly the same results (albeit, computed much faster). See figure 3

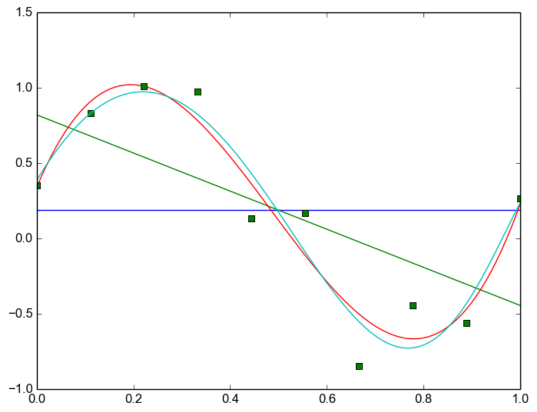

Figure 2: Plot of weighted polynomials, computed with gradient descent, step size 0.01, threshold 10^{-7}, initial guess, weight 1 for all dimensions
Blue: Order 0, Green: Order 1, Red: Order 3, Cyan: Order 9

It is unsurprising that both of these gradient descent methods converge in the $M = 9$ case to the state they do. Both $M = 9$ shapes are similar to the $M = 3$ shape and reasonabily approximate the original generating function. This position is a local minimum and finding the true global minimum would require more sophisticated techniques.

2.4 Sinusoidal Basis

We expect that using sinusoidal basis functions would fit this data very easily as $\sin(2\pi x)$ is the true generating function and a weight of 1 on this term would establish a strong local minimum. Sinusoidal basis

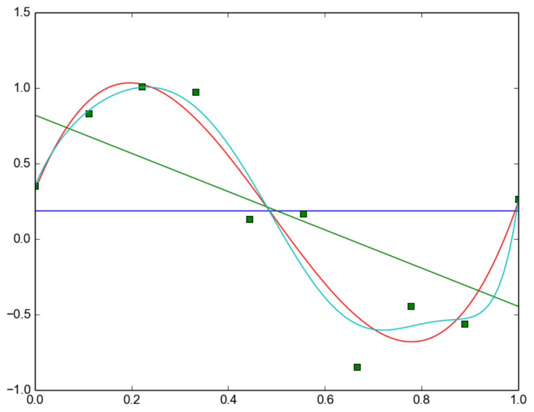

Figure 3: Plot of weighted polynomials, computed with builtin fmin_bfgs
Blue: Order 0, Green: Order 1, Red: Order 3, Cyan: Order 9

functions will work well for periodic data, however, if the input is non-periodic (e.g $y = x$) the weights will correspond to the Fourier series of the ground truth. This will require a high dimensionality to accurately fit the data.

3 Ridge Regression

3.1 Ridge Regression

We used the closed form optimal solution of ridge regression to calculate the weights. The equation is given by $w = (\lambda I + \Phi^T \Phi)^{-1} \Phi^T Y$. Next, we iterated through different values of lambda and order.

Increasing λ for a given value of M makes the curve flatter and less bumpy. For a λ value high enough, the curve becomes horizontal, since a non-zero weight would incur a large penalty from the regularization term. The flattening curve can best be seen in the M=1 case (Fig 4) where it is clear that the line corresponding to higher λs are flatter. On the other hand, with a $\lambda = 0$, the regularization term drops out and it becomes a normal gradient descent. In this case, the curves we generated exactly matched the curves from Bishop Fig. 1.4. Increasing M for a given value of λ increases the fit and makes the curve bumpier. With a $\lambda = 0$ and $M = 9$, the resulting polynomial perfectly fits the 10 data points in Bishop.

3.2 Model Selection

Order	λ	**Validation**	**Test**
0	10^{-5}	15.77	19.57
0	1	14.41	18.47
0	10^{5}	12.62	19.21
2	10^{-5}	4.74	11.43
2	1	4.50	11.77
2	10^{5}	12.61	19.21
4	10^{-5}	5.40	22.96
4	1	0.53	13.49
4	10^{5}	12.60	19.46

We looked at all combinations of order and λ for order between 0 and 5 and log λ between -5 and 4. We used the training dataset to optimize the parameters, and then used the SSE on the validation dataset to compare parameters. The combination that minimizes validation error is $M = 4$ and $\lambda = 1$ for the values

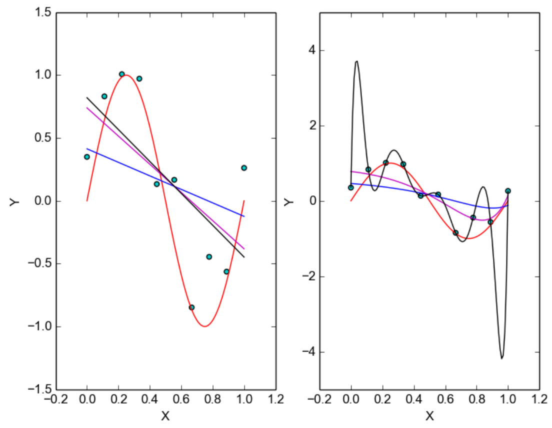

Figure 4: Plot of weighted polynomials, computed using ridge regression
Order: Left=1 Right=9
Λ: Blue=1.0 Magenta=0.1 Black=0.0

we chose. This gives a test error of 13.49. The parameters that minimize the test error, however, is $M = 1$ and $\lambda = 1$.

3.3 BlogFeedback Dataset

We now use a similar procedure on the larger BlogFeedback dataset, but we substitute the polynomial basis for a simple linear model. In addition, we scale the features by the second norm. We iterate over values of λ with a range of -10 to 100, and find that the minimizing value is 11, with a sum of squared errors on test error equal to $5.59 * 10^6$. Figure 5 shows that the error drops off rapidly and flattens around 11.

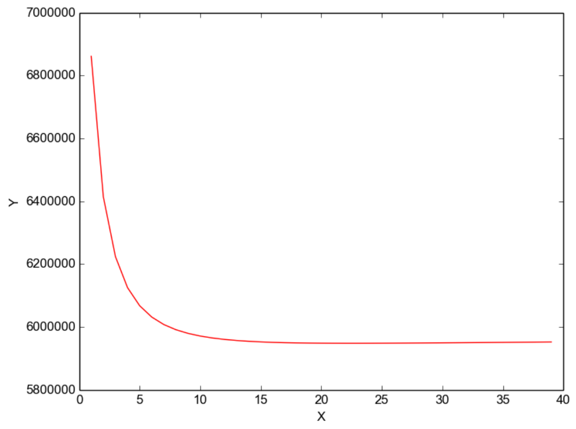

Figure 5: Sum of Squared Errors as function of λ

4 Outlier and LAD

The equation we seek to minimize in Least Absolute Deviations is

$$\|y - \Phi w\|_1 + \lambda \|w\|_2$$

, where we take the first norm for the errors and the second norm for the regularization term. Instead of finding a closed form expression for the derivative, we used gradient descent on the above error function with the built-in *fmin_bfgs* minimizer. In order to find good model parameters M and λ, we iterate through the same values as before, and pick M and λ to minimize the sum of squared errors on the validation data set after estimating the weights on the training dataset. Of the values we surveyed, $M = 4$ and $\lambda = 1$ minimizes the sum squared error, which is 0.3898. The resulting curve is in Figure 6.

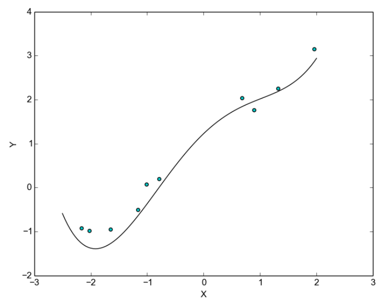

Figure 6: $M = 4$ and $\lambda = 1$

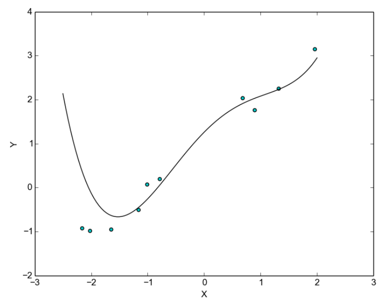

Figure 7: $M = 4$ and $\lambda = 0.1$

When the λ is low and the order is high, the model tries to fit an outlier in the training data located close to $(-2, 2)$. Figure 7 shows the curve with $\lambda = 0.1$. However, as λ increases, the curve flattens out and ignores the outlier, as is the case in Figure 6. Conversely, as the order rises, the curve begins to overfit even for higher values of λ, as is evident in figure 8. By contrast, for lower orders, even smaller values of λ do not run into over-fitting issues.

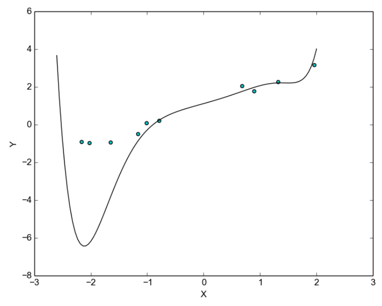

Figure 8: $M = 9$ and $\lambda = 1$

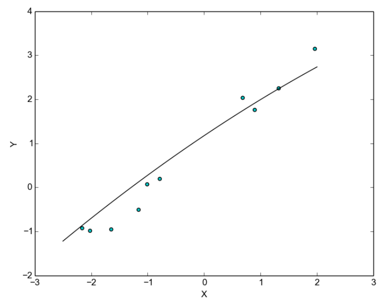

Figure 9: $M = 2$ and $\lambda = 0.01$

Because ridge regression takes the second norm of the difference between actual and predicted values, differences between outliers and the fitted curve will be magnified by the square. LAD performs better when many outliers are present in the data, since it only takes an absolute value. On the Bishop data, even with $M = 9$ and $\lambda = 0.1$, the LAD regression did not fit every single point like the ridge regression did. Instead, it fit a relatively flatter curve.