Anonymous Authors
September 29, 2017

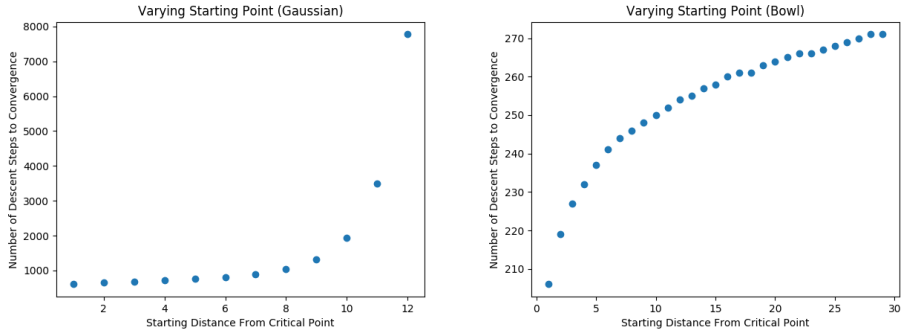
6.867 Homework 1

1 Implementing Gradient Descent

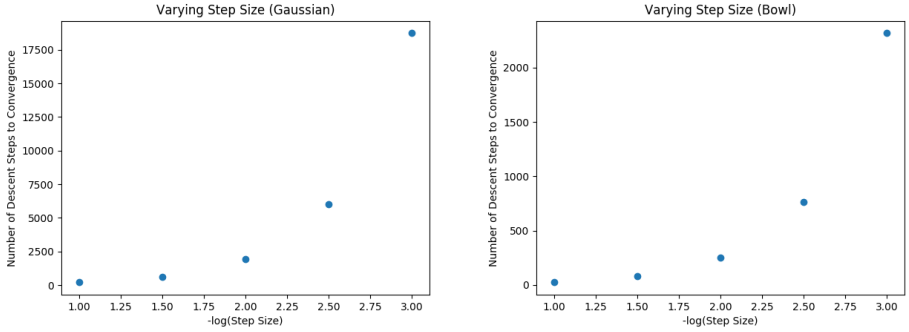
In any gradient descent algorithm, the main hyperparameters we have to tune are the initial point we start the gradient descent from, the step size, and the convergence criteria.

For each of the three parameters, we can see how varying the parameters changes the gradient descent for both of the provided functions (the Gaussian and the bowl).

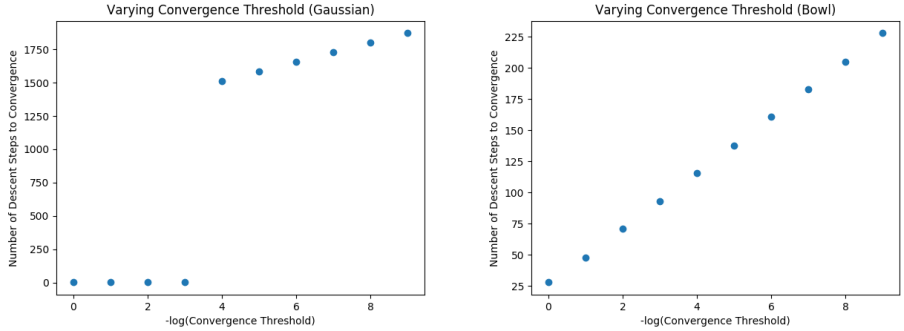
Unless otherwise specified, default parameters are: starting point = (0,0), step size = 0.01, and convergence criteria of difference between consecutive objective function values is less than 10^{-10}.



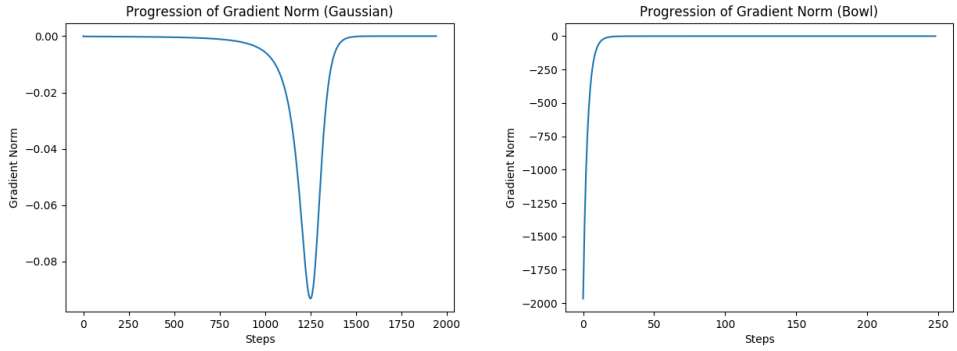
The number of steps needed for convergence grows much faster in the Gaussian than it does with the bowl. This is because the gradient decays exponentially in the Gaussian case with respect to distance from the critical point but linearly in the bowl case.



In both cases, step size eventually grows linearly with the number of steps needed until convergence. However, in some cases where the step size is extremely large (not shown), the function may not converge at all.

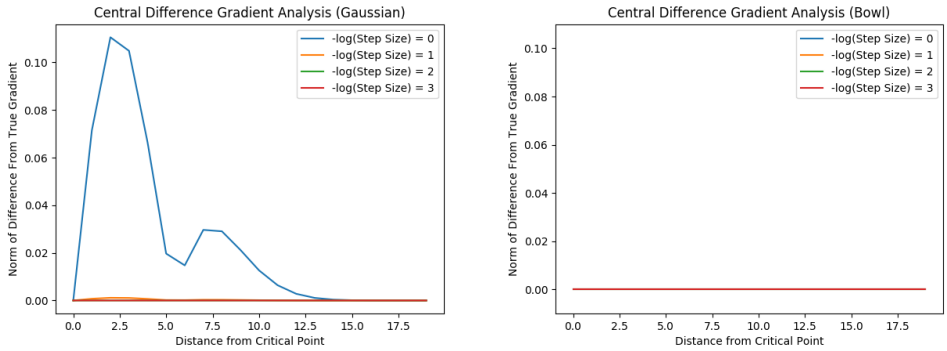


Here, as the convergence threshold grows, so to does the number of steps needed to converge. Intuitively, this makes sense–the more accurate the gradient descent needs to be, the more iterations we need to converge.



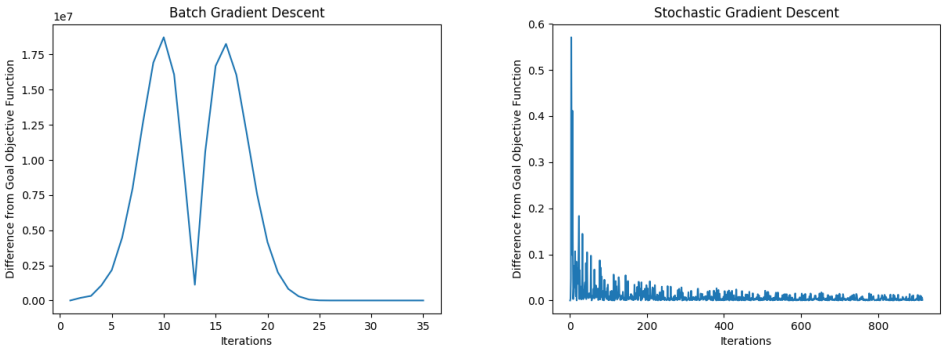
As mentioned before, the gradient norm of the Gaussian starts off very small when the starting point is far away from the critical point. Around step 1000, it starts to reach the steep part of the Gaussian, at which point it quickly reaches convergence. For the bowl, the gradient norm starts off high and quickly decreases and converges.

To perform gradient descent on functions that do not have a clean, closed-form gradient, we can approximate the gradient using central differences. Here, we analyze how the performance of the central differences approximation varies based on the step size used for the difference approximation.



When the step size is larger than 10^{-2}, we start to see errors in the Gaussian. For the bowl, we have no error in our gradient approximation regardless of step size because the gradient of the bowl is linear; because of this, our averaging approximation acutally gives us the exact answer.

Often, when running gradient descent, it is computationally expensive to calculate the total error. We can use Stochastic Gradient Descent as an alternative: each time we calculate the least square error, we only use one sample to calculate the error. This results in a much faster overall computation at the expense of some accuracy. We can compare the two algorithms by calculating the gradient descent of the least square error with similar convergence criteria.



It takes many more iterations for SGD to converge than batch descent. Furthermore, batch descent can converge on a more accurate answer (due to the randomness of SGD throwing things off). However, SGD runs much faster, even while taking more iterations. In practice, mini-batch SGD is often used as the best of both worlds.

2 Linear Basis Function Regression

2.1 Closed-form Solution with Polynomial Basis

Consider the basis function

$$\mathbf{\Phi}_M(x) = [\phi_0(x), \ldots, \phi_M(x)],$$

where each $\phi_k(x) = x^k$. Applying $\mathbf{\Phi}_M$ to $\mathbf{X}$ gives the desired basis change, so we have the generalized linear model

$$\mathbf{Y} = \mathbf{\Phi}_M(\mathbf{X}) \cdot \boldsymbol{\theta} + \boldsymbol{\epsilon},$$

which has the close-form solution (from first order conditions on the least squares error[1]),

$$\hat{\boldsymbol{\theta}} = (\mathbf{\Phi}_M(\mathbf{X})^T \mathbf{\Phi}_M(\mathbf{X}))^{-1} \mathbf{\Phi}_M(\mathbf{X})^T \mathbf{Y}$$

where $\hat{\boldsymbol{\theta}}$ is the maximum-likelihood estimator of the regression coefficients.

We fit polynomial regressions on the data via this method with a few different max degrees ($M = 0, 1, 3, 10$). Below are the resulting polynomials, plotted against the original data and source function:

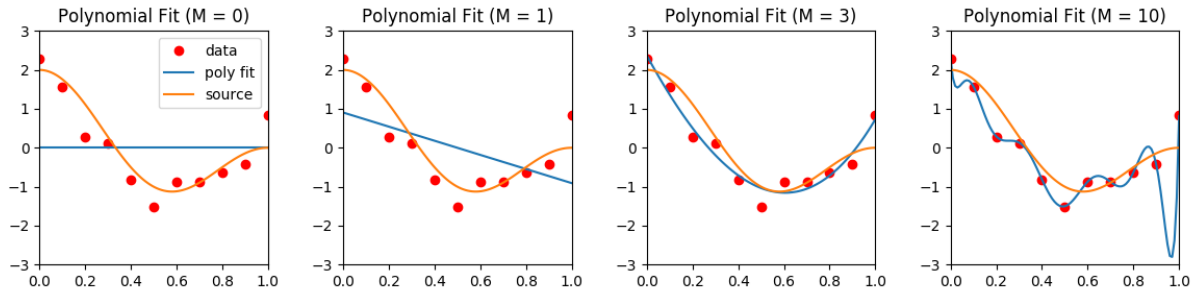


Figure 7: M-Degree Polynomial Fit

It appears that $M = 3$ gives the closest interpolation of the true function. We infer that $M = 0, 1$ give a poor fit as the models do not have sufficient degrees of freedom, while $M = 10$ has too many paramters and leads to overfitting.

[1] See section 2.2 for a derivation of the gradient vector.

2.2 Closed-form Gradient of Squared Error

We take a second approach to finding the least squares estimators (which also happen to be maximum likelihood estimators). The residual sum of squares for weight vector θ is given by

$$loss(\boldsymbol{\theta}) = |(\boldsymbol{Y} - \boldsymbol{\Phi}_M(\boldsymbol{X})\boldsymbol{\theta})|^2.$$

Differentiating the error function with respect to $\boldsymbol{\theta}$ gives the gradient function

$$\begin{aligned}
\frac{\partial}{\partial \boldsymbol{\theta}}|(\boldsymbol{Y} - \boldsymbol{\Phi}_M(\boldsymbol{X})\boldsymbol{\theta})|^2 &= \frac{\partial}{\partial \boldsymbol{\theta}}[(\boldsymbol{Y} - \boldsymbol{\Phi}_M(\boldsymbol{X})\boldsymbol{\theta})^T \cdot (\boldsymbol{Y} - \boldsymbol{\Phi}_M(\boldsymbol{X})\boldsymbol{\theta})] \\
&= \frac{\partial}{\partial \boldsymbol{\theta}}[-(\boldsymbol{\Phi}_M(\boldsymbol{X})\boldsymbol{\theta})^T\boldsymbol{Y} - \boldsymbol{Y}^T(\boldsymbol{\Phi}_M(\boldsymbol{X})\boldsymbol{\theta}) + \boldsymbol{\theta}^T\boldsymbol{\Phi}_M(\boldsymbol{X})^T \cdot (\boldsymbol{\Phi}_M(\boldsymbol{X})\boldsymbol{\theta})] \\
&= -2 \cdot \boldsymbol{\Phi}_M(\boldsymbol{X})^T\boldsymbol{Y} + 2 \cdot \boldsymbol{\Phi}_M(\boldsymbol{X})^T\boldsymbol{\Phi}_M(\boldsymbol{X})\boldsymbol{\theta}
\end{aligned}$$

Using the numerical derivative code in Section 1, we tested the correctness of the closed-form gradient on a few different $\boldsymbol{\theta}, M$ values, and the results are all accurate within 10^{-10} of each other.[2]

2.3 Gradient Descent Solution with Polynomial Basis

We run batch and stochastic gradient descent on the square loss, and experimented with differents start points, convergence thresholds, and learning rates (i.e. step sizes). We find that the smaller convergence thresholds produce slightly better fits, which is expected, but the effect is not significant. During our experimentation, convergence thresholds smaller than 10^7 generally produced the same weights.

We also find higher degree models are more sensitive to the start point than the low degree models. This is likely because models with a higher degree M have more local minima that the algorithm may converge to.

In addition, we find that the solutions are quite sensitive to the learning rates, and that SGD tend to require a higher learning rate than Batch in general to achieve a reasonable fit. Since the gradient vector at each step in SGD much noiser compared to in Batch, SGD has is more likely to descend into some spurious equilibrium. This can be partially compensated by a higher learning rate. (However, a learning rate too large causes the the algorithm to skip over the global minima to fail to converge.)

By similar reasoning, we find that Batch produces a much better fit than SGD, especially given the small data set (so runtime is not an issue). This is illustrated by plots of Batch and SGD below (start point = $\mathbf{0}$, convergence threshold = 10^{-8}):

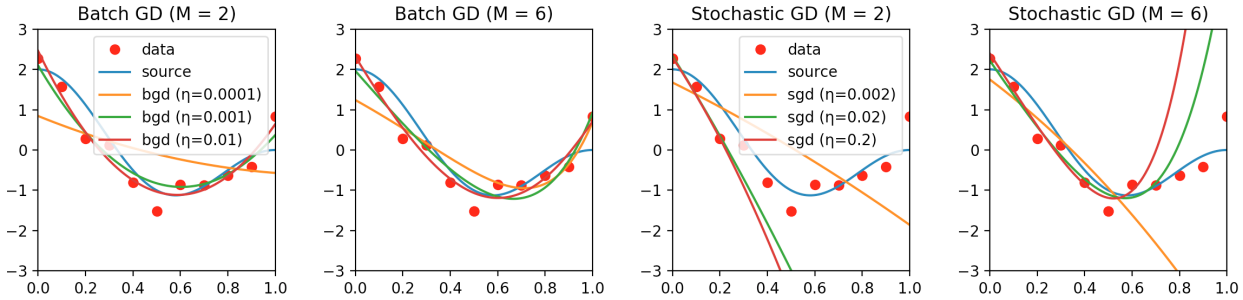


Figure 8: Batch and Stochastic Gradient Descent with Various Learning Rates η

2.4 Closed-form Solution with Cosine Function Basis

We apply the same technique as in section 2.1, but now with cosine functions as basis. We apply the basis transformation of degree M

$$\boldsymbol{\Phi}_M(x) = [\phi_1(x), \ldots, \phi_M(x)],$$

[2]Measured in the squared norm of the difference.

where each $\phi_k(x) = cos(k\pi x)$, and solve for closed-form solutions to the generalized linear model of degrees M = 1, 2, 4, 8. We plot the results against the data and source below:

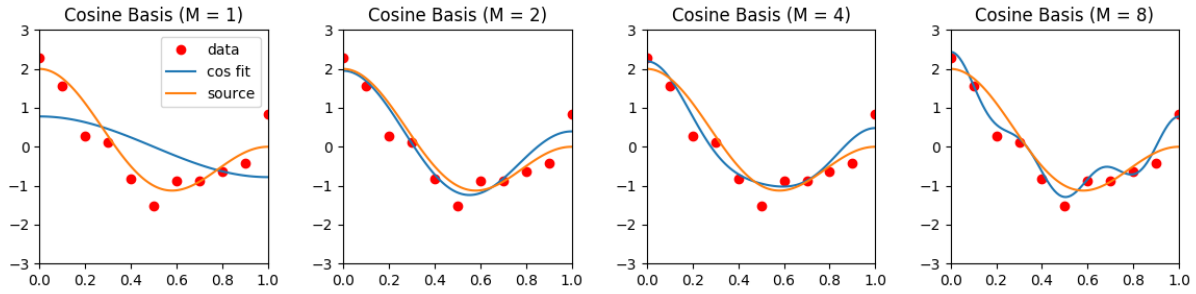


Figure 9: Cosine Functions Fit

Again, $M = 2$ seems to give the best fit of the true function, and overfitting behavior is clear when the degree is high (such as in $M = 8$).

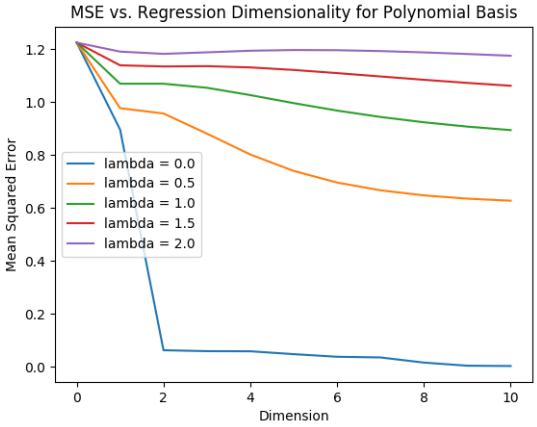
For $M = 2$, the regression result is $f(x) = 0.78\cos(\pi x) + 1.17\cos(2\pi x)$, which resembles the true function pretty well for most values of x, as seen in the plot. However, the coefficient estimates are still off by around 20%, which is not surprising given the very limited amount of data.

3 Ridge Regression

We first run ridge regression on the polynomial basis data set from the previous problem. Given that we are trying to minimize the squared error $||X\theta - y||^2 + \lambda||\theta||^2$, the closed form solution gives us:

$$\theta = (X^T X + \lambda I)^{-1} X^T y$$

We plot the MSE for different dimensionality models for different values of λ.



For $\lambda = 0$, increasing the dimensionality slowly improves the MSE. However, as λ grows, the benefit shrinks because the regularization term punishes large terms in θ.

We then run our ridge regression on the provided datasets. For each value of λ, we run ridge regression for different values of M and pick the model with the best validation error. We then run that model on the test set and compute the MSE. However, if we just run the training on A and validation on B, we get large MSEs for all values of λ because there is one outlier point in data set B: (-2.60,3.46). These results look weird

because the test errors are so much higher than the validation errors. This is because there is a clear outlier data point in B that throws off the MSE.

Two ways to deal with outliers include 1) upper-bounding errors incurred by single points, or 2) removing the points entirely. We experimented with removing the outlier point and ran our ridge regression again.

Train on A, Test on Modified B			
λ	Optimal M	Validation MSE	Test MSE
0	2	0.107	0.06
0.5	2	0.126	0.08
1	2	0.147	0.106
1.5	2	0.170	0.133
2	1	0.188	0.158
2.5	1	0.206	0.183
3	1	0.225	0.209
3.5	1	0.245	0.235
4	1	0.265	0.262

Train on Modified B, Test on A			
λ	Optimal M	Validation MSE	Test MSE
0	3	0.089	0.193
0.5	4	0.091	0.084
1	1	0.125	0.079
1.5	1	0.132	0.077
2	1	0.144	0.078
2.5	1	0.159	0.084
3	1	0.177	0.092
3.5	1	0.197	0.102
4	1	0.218	0.115

We see some differences between the two test runs. When training on A and testing on B, the best model is quadratic, while it is linear when swapping the training and test sets. In both cases, though, the models that performed well had relatively low dimension, which makes sense given what the data looks like. Furthermore, setting λ above 3 seemed too harsh of a regularization penalty. Ultimately, given the provided data sets, picking a linear or quadratic model with a small regularization penalty seems like the best bet.

4 Sparsity and Lasso

We use the *sklearn* package in Python to run lasso regression. We run lasso on the training set with different values for the parameter λ. Next, we test the trained models on the validation set, and find that the model trained with parameter $\lambda = 9.4 \cdot 10^{-4}$ gives the least square loss in the validation set. We observe that for sufficiently large $\lambda \approx 10^{-3}$, we start to get sparse solutions from lasso, with nonzero coefficients in the second, third, fifth, and sixth terms (which matches those of the true weight vector). Lastly, we find the residual of sum of squares of estimated weights in the test data to check if we overfit.

We run the same procedure with ridge regression. In comparison, ridge regression does not give a sparse representation, as all thirteen weights are nonzero.

We have residual sum of squares $(RSS_{val}, RSS_{test}) = (0.51, 0.74)$ for lasso, and $(RSS_{val}, RSS_{test}) = (0.46, 0.85)$ for ridge regression. Both lasso and ridge regression performed very well in fitting the data, with slightly more overfitting in ridge regression than in lasso (as seen from the larger test error). It is not surprsing that lasso performed better, as the true function only has four parameters as well.

See plot below for comparison of lasso, ridge regression, ordinary least squares with basis change, and the true function:

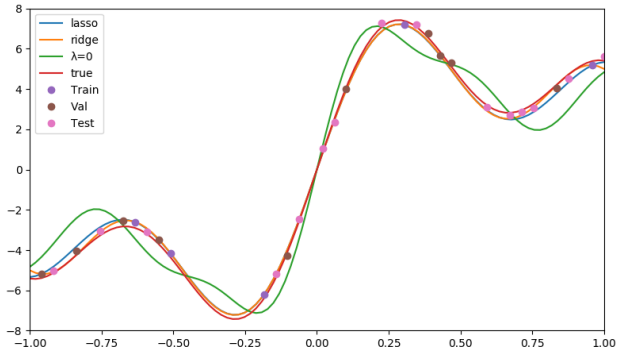


Figure 10: Lasso vs. Ridge regression