# 6.867: Problem Set 1

Rachel Wu, Tony Zhang

September 29, 2016

## 1 Gradient Descent

### 1.1 Batch Gradient Descent

We implemented a general-purpose batch gradient descent procedure, as described in Bishop 5.2.4, with user-specified objective function and gradient thereof, initial guess, step size, and termination criterion. We elected to terminate once the objective function change fell below a given threshold. We tested our implementation against two functions: a sign-reversed multivariate Gaussian with mean $\mu$ and covariance matrix $\Sigma$

$$f(x;\mu,\Sigma) = -\frac{e^{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)}}{\sqrt{(2\pi)^n \det \Sigma}} \tag{1}$$

and a quadratic bowl

$$f(x;A,b) = \frac{1}{2}x^TAx - x^Tb, \tag{2}$$

where $A$ is positive-definite. For this exercise, our Gaussian parameters were $\mu = (10,10)^T, \Sigma = 1000 \cdot I_2$. For the quadratic bowl, we took

$$A = \begin{pmatrix} 10 & 5 \\ 5 & 10 \end{pmatrix}$$

and $b = (400,400)^T$. We present our results in Figure 1, where we illustrate the effect of the initial guess and step size for convergence on these two objective functions.

### 1.2 Numerical Gradient Approximation

Recall that our gradient descent procedure requires the user to specify both the objective function *and* its gradient. In our examples, these gradients had simple analytic forms, but in general objective functions might have very complicated gradients. Therefore, as a check on (or as a substitute for) the user-specified gradient, we implemented a gradient approximation routine.

We approximate the gradient of a general function $f : \mathbb{R}^d \to \mathbb{R}$ at given point $x^*$ by approximating its partial derivatives as the central difference

$$\frac{\partial f}{\partial x_i}(x^*) \approx \frac{f(x^* + \frac{1}{2}\varepsilon\hat{x}_i) - f(x^* - \frac{1}{2}\varepsilon\hat{x}_i)}{\varepsilon}, \tag{3}$$

where $\varepsilon > 0$ is some small user-supplied difference step and $\hat{x}_i$ is the unit vector in the $x_i$ direction. We compute this approximate partial derivative along each dimension of our domain and collect the terms into our numerical gradient approximation.

Using our approximation procedure, we verified the analytic gradient calculations we made for the negative Gaussian and the quadratic bowl. We obtained the best approximations when $\varepsilon$ was on the order of the radius of curvature at the point of interest (indeed, tighter structure warrants a smaller $\varepsilon$). Too small an $\varepsilon$ resulted in rounding errors and other numerical instability, while too large an $\varepsilon$ overstretched the secant approximation our procedure relies on.

### 1.3 Stochastic Gradient Descent

By design, batch gradient descent determines each update step by looking at the entire dataset and taking a gradient. This approach becomes unfeasible with large datasets, and does not extend easily to online settings. For these reasons, we instead use stochastic gradient descent (SGD), in which we approximate the gradient as follows.

Suppose our objective function takes the form

$$J(\theta) = \sum_i J_i(\theta) \tag{4}$$

where each $J_i$ is presumably a term corresponding to a single data point. Whereas the batch gradient descent update step takes a gradient over all $J$:

$$\theta_{t+1} = \theta_t - \eta\nabla J(\theta_t), \tag{5}$$

the SGD update step has the form

$$\theta_{t+1} = \theta_t - \eta_t\nabla J_i(\theta_t). \tag{6}$$

Note two key differences. We take $\nabla J_i$ as an approximation of $\nabla J$. More interestingly, our step size is now time-dependent. Because SGD is stochastic, it is unclear whether it converges. But it can be shown that choosing a learning rate schedule $\eta_t$ satisfying the Robbins-Monro conditions ($\sum_{t=1}^{\infty} \eta_t$ diverges and $\sum_{t=1}^{\infty} \eta_t^2$ converges) guarantees convergence.

We implemented SGD with two stopping criteria: either gradient norm or change in objective function drop below a provided threshold. In practice, we chose to set a threshold on change in objective function, since it is more readily interpretable.

We tested both batch and stochastic gradient descent on a ordinary least-squares fitting problem, where we

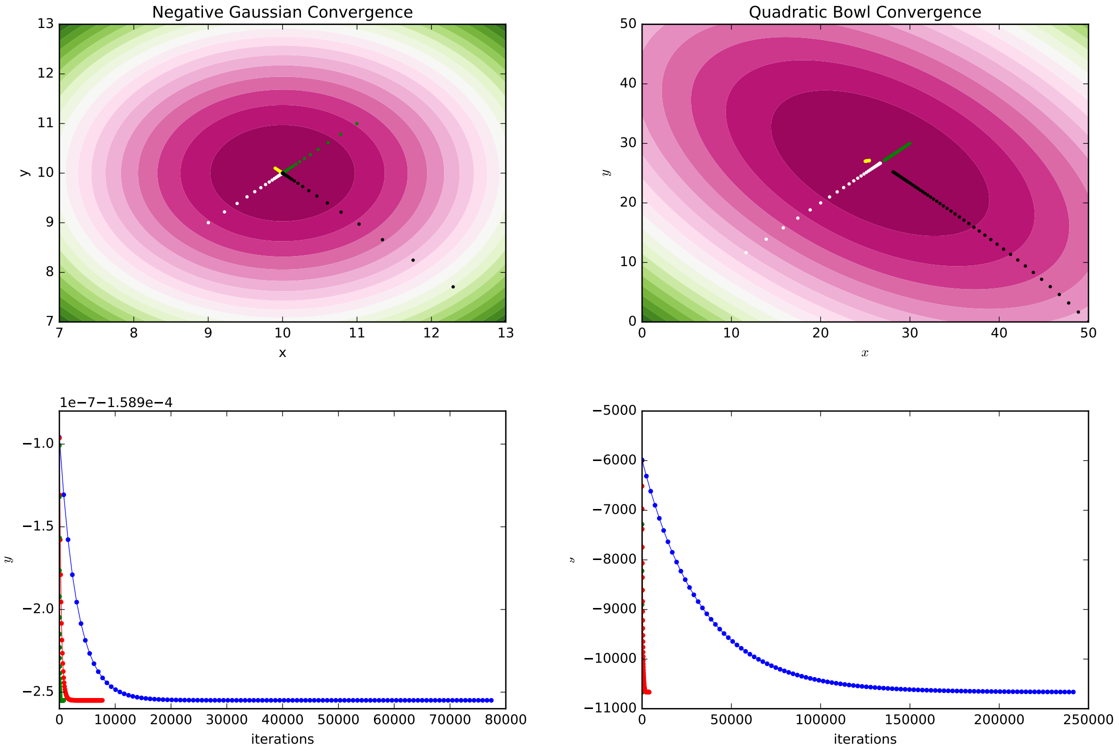

Figure 1: Effects of initial guess and step size on convergence. Bad initial guesses took much longer to converge (8428 and 5743 iterations) than good initial guesses (6292 and 372 iterations, respectively). The bottom two figures show the change in objective function over time. Larger step sizes (red and green) converged faster, if at all.

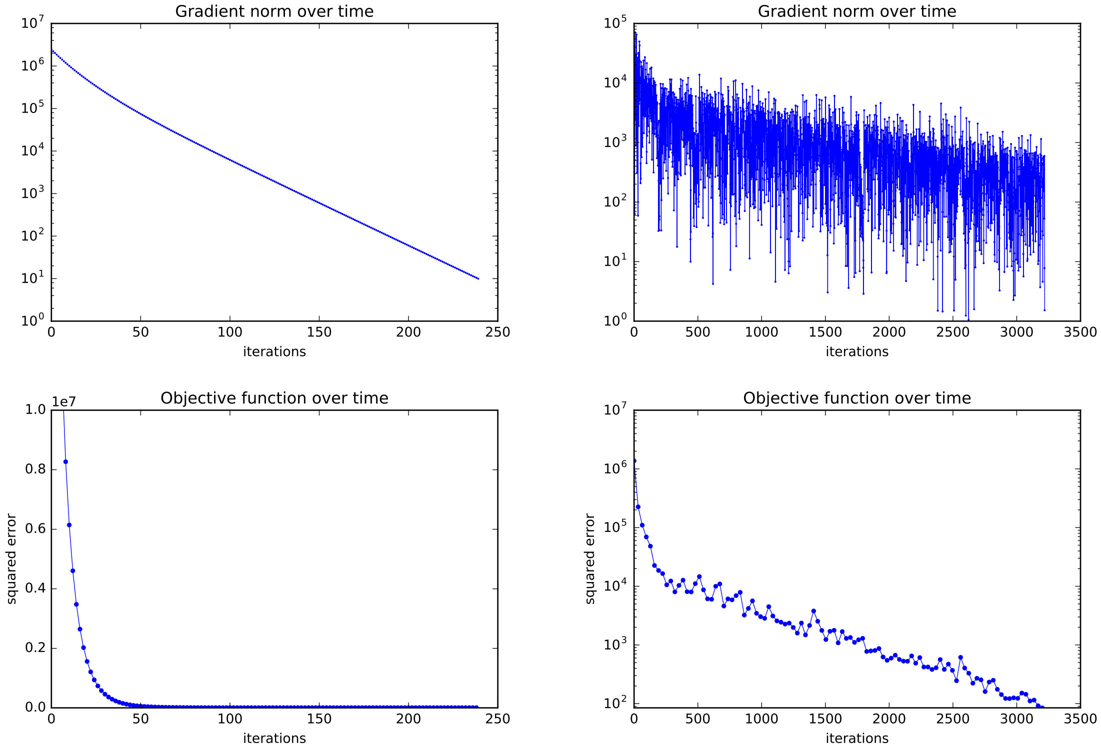

Figure 2: Batch gradient descent (left) and stochastic gradient descent (right) on the sum of squares error for linear regression on 100 data points in $(\mathbb{R}^{10}, \mathbb{R})$. Observe that SGD requires more iterations to converge and fluctuates more, in both objective function and gradient, than batch gradient descent.

minimized the sum of squares error. We present the results of both tests in Figure 2. Note that while batch gradient descent converges smoothly, the loss function fluctuates as SGD runs. In both cases, we eventually get convergence.

# 2 Linear Regression

Before proceeding, recall that a linear basis function model for a dataset $(x^{(n)}, y^{(n)})$ given a fixed set of basis functions $\phi_i$ is a regression model that tries to find weights $w_i$ for which we approximately have

$$y^{(n)} = w_0 + \sum_i w_i \phi_i(x^{(n)}). \tag{7}$$

In other words, we transform the independent variable $x$ by basis functions $\phi_i$ into some feature space, then perform linear regression with independent variables $\phi_i(x)$ and dependent variable $y$.

We wrote a procedure for general linear basis function regression with user-specified data points and basis functions that found the maximum-likelihood weight vector analytically, via Bishop Equation 3.15, which we restate here:

$$w_{\text{ml}} = (\Phi^T \Phi)^{-1} \Phi^T Y \tag{8}$$

where $Y$ is a vector of $y^{(n)}$ values from our dataset and $\Phi$ is the design matrix, given by

$$\Phi_{ij} = \phi_j(x^{(i)}). \tag{9}$$

## 2.1 Polynomial Basis Functions

We tested our procedure by performing fits on a dataset of 11 points with polynomial basis functions up to some specified degree $M$. The dataset was generated by applying noise to the values produced by the function

$$y(x) = \cos(\pi x) + \cos(2\pi x). \tag{10}$$

We plot the results of these fits for varying values of degree $M$ in Figure 3. Note the obvious underfitting and overfitting introduced by models with excessively low or high degrees.

## 2.2 Regression by Gradient Descent

Though there is a closed-form formula for the best-fit weight vector (Equation 8), we decided to verify our results from the previous section through gradient descent.

To this end, we considered the sum-of-squares error (SSE) for our problem:

$$\begin{aligned} J(w) &= \frac{1}{2} \sum_n \left( y^{(n)} - \sum_j \phi_j(x^{(n)}) w_j \right)^2 \\ &= \frac{1}{2} (\Phi w - Y)^T (\Phi w - Y), \end{aligned} \tag{11}$$

which has gradient

$$\nabla J(w) = \Phi^T (\Phi w - Y). \tag{12}$$

We verified the correctness of Equation 12 with our numeric gradient approximation from section 1.2.

We proceeded to estimate the maximum-likelihood weight vector $w_{\text{ml}}$ (Equation 8), which minimizes $J(w)$, using batch gradient descent on the SSE. For small degree fits, results agreed strongly with the closed-form solution. However, for higher degrees, discrepancies grew between our maximum-likelihood fits and our gradient-descent estimate fits (Figure 4).

Interestingly, the gradient descent fits did not overfit nearly as much as the maximum-likelihood fits at high degree. This behavior likely arises because we initialize our weight vectors at 0, "regularizing" our fits in a sense. Indeed, initializing our weight vectors to very large (in absolute value) random numbers produced very high-variance fits.

Moreover, our gradient descent procedure terminates when the objective function changes between gradient descent iterations drops below a certain threshold, which we can think of as a form of early stopping. Intuitively, overfitting requires a large change in our model parameters to get small decreases in our loss function; our convergence criterion prevents this sort of micro-optimization from occurring.

Varying the step size and the convergence threshold had little effect at any degree, unless we changed the parameters by many orders of magnitude. (e.g. large step sizes can lead to divergence)

We repeated this analysis with stochastic gradient descent (SGD), which was much more sensitive to our initial guess. Unless we initialized our weights closed to the correct (maximum-likelihood) value, SGD produced very poor fits, even at low degree. At higher degrees, even picking initial weights close to their correct value did not seem to help with convergence.

This behavior can be explained if we consider the size of our dataset. With a small dataset, it is more likely that the algorithm will believe it has converged when it hasn't, since it is conceivable that we terminate after checking a point that is already fit well and noticing that the resulting weight update changes the objective function very little.

## 2.3 Cosine Basis Functions

We also experimented with using cosine basis functions $\phi_m(x) = \cos(m\pi x)$ to fit the dataset. Recall that the values were generated from Equation 10, so we expect our weights to be $(0, 1, 1, 0, \dots)$ (recall that the first term is the weight for $\phi_0(x) \equiv 1$, so it is a bias term). With a fit degree of 8, however, we overfitted significantly, and our weight vector differed greatly from the actual weights used to generate the data. In section 4 we will enforce sparsity on our linear models to produce better fits in certain scenarios.

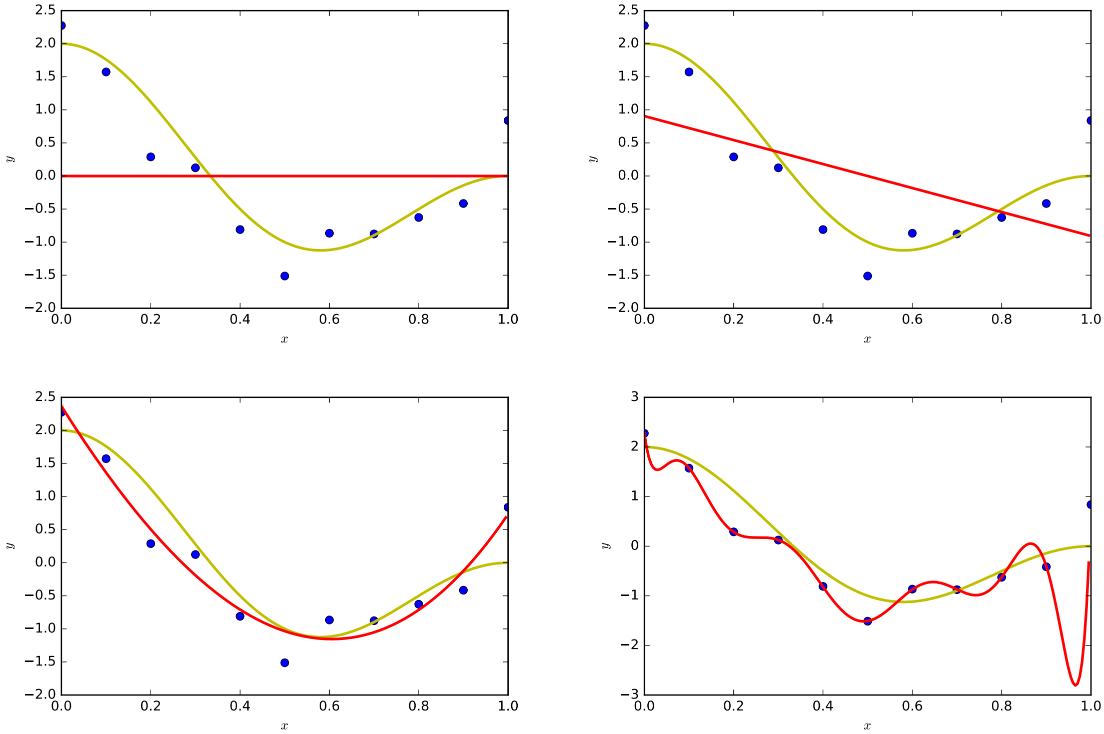

Figure 3: Plots of our polynomial basis fits for degrees $M = 0, 1, 3, 10$ (red) of our dataset (blue), which was generated from an underlying function (yellow). The degree ascends from top to bottom, and within each row, from left to right.

# 3 Ridge Regression

Recall that ridge regression is just linear regression, with an $L^2$ regularization term added to the loss function. That is, our error is the following modification of Equation 11 (Bishop Equation 3.27):

$$J(w) = \frac{1}{2}(\Phi w - Y)^T(\Phi w - Y) + \frac{1}{2}\lambda |w_+|^2, \tag{13}$$

where $w_+$ is the weight vector $w$ *without* the bias term and $\lambda \geq 0$ controls the strength of regularization. The extra term penalizes large weights.

Then it can be shown that the optimal weight vector is given by

$$w_{+,\text{ridge}} = (\lambda I + Z^T Z)^{-1} Z^T Y_c \tag{14}$$

where $Z = \Phi - \overline{\Phi}$, where $\Phi$ is taken with *no* column of 1's, and where $\overline{\Phi}$ is the average of all rows of $\Phi$ (an average of data points); and where $Y_c$ is $Y - \bar{Y}$, where $\bar{Y}$ is the mean of $Y$; and bias term

$$w_{0,\text{ridge}} = \overline{Y} - w_{+,\text{ridge}}^T \overline{\Phi} \tag{15}$$

We implemented ridge regression and tested it on the data from our previous experiments on regression for a variety of polynomial degrees and values of $\lambda$. We present some of the fits in Figure 5.

Note that when fitting with higher order polynomial bases, even small values of $\lambda$ can drastically reduce overfitting. In general, at all degrees, larger values of $\lambda$ "smooth" out and slightly "flatten" our fit. Indeed, in the linear case, larger values of $\lambda$ would decrease the weight, i.e. the slope of our regression line.

## 3.1 Model Selection

We now turn our attention to another dataset, again of pairs of real numbers. This dataset has been partitioned into sets $A$, $B$, and $V$, with respective sizes 13, 10, and 22, and is plotted in Figure 6. We will use $V$ as our validation set. We will try to model this data with a polynomial fit, as before, except we will be using regularization.

We used $A$ as training data and $B$ as test data, then vice versa. In each case, we found the optimal weights by Equations 14 and 15 on training data for many values of the degree $M$ of the polynomial basis and the regularization parameter $\lambda$. We chose the $(M^*, \lambda^*)$ that produced the lowest *validation* error, then evaluated the model on test data. Note that we evaluate performance of an actual weight vector with ordinary mean-square error (MSE; Equation 11 normalized by the number of data points), with *no regularization term*.

We present the validation errors we obtained in Tables 1 and 2. When we trained on $A$ and tested on $B$, the best hyperparameters were $(M^*, \lambda^*) = (2, 0)$; the validation MSE was 0.05336, and the test MSE was 1.28765. When we trained on $B$ and tested on $A$, the best hyperparameters $(M^*, \lambda^*) = (3, 0.6)$ produced a validation MSE of 0.65195 and a test MSE of 1.51697. We plot these two best-fit models in Figure 6.

Before concluding, observe from Tables 1 and 2 that

Table 1: Mean-square error on our validation set after training on $A$. The optimal pair of hyperparameters was $(M^*, \lambda^*) = (2, 0)$.

| $M$<br>$\lambda$ | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | 1.40909 | 0.06584 | 0.05336 | 0.05386 | 0.08134 | 0.09403 |
| 0.001 | ″ | 0.06584 | 0.05336 | 0.05388 | 0.08129 | 0.09393 |
| 0.003 | ″ | 0.06585 | 0.05337 | 0.05392 | 0.08119 | 0.09373 |
| 0.01 | ″ | 0.06587 | 0.05339 | 0.05407 | 0.08086 | 0.09305 |
| 0.03 | ″ | 0.06596 | 0.05345 | 0.05450 | 0.07998 | 0.09120 |
| 0.1 | ″ | 0.06624 | 0.05366 | 0.05600 | 0.07747 | 0.08564 |

Table 2: Analogous to Table 1, except with training on $B$. The best hyperparameters are $(M^*, \lambda^*) = (3, 0.6)$.

| $M$<br>$\lambda$ | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | 1.64635 | 0.79978 | 0.87563 | 0.68864 | 2.95901 | 14.21443 |
| 0.03 | ″ | 0.80066 | 0.87533 | 0.68262 | 2.78933 | 13.10785 |
| 0.1 | ″ | 0.80271 | 0.87471 | 0.67110 | 2.46809 | 11.18677 |
| 0.3 | ″ | 0.80852 | 0.87347 | 0.65314 | 1.90417 | 8.20467 |
| 0.6 | ″ | 0.81709 | 0.87289 | 0.65195 | 1.51048 | 6.18385 |
| 1.0 | ″ | 0.82828 | 0.87410 | 0.67289 | 1.29261 | 4.82707 |

increasing the degrees of freedom in our model increases the optimal $\lambda$. Indeed, more degrees of freedom require more regularization to prevent overfitting.

## 4 Sparsification with LASSO

In section 2.3, recall that we attempted a cosine basis fit of data generated by Equation 10. The model overfit the data significantly, as was apparent graphically, so we might therefore consider regularization. Instead of ridge regression (quadratic regularization), however, we used the LASSO (least absolute shrinkage and selection operation), which enforces our prior beliefs about the sparsity of the weight vectors.

Recall that in ridge regression, we modified our loss function to include a quadratic regularizer. We can generalize Equation 13 (Bishop 3.29) to other exponents. In particular, when the exponent is 1, we have the LASSO error function:

$$J(w) = \frac{1}{n} \sum_{i=1}^{n} (y^{(i)} - w^T \phi(x^{(i)}))^2 + \lambda \sum_{j=1}^{M} |w_j|. \tag{16}$$

We evaluated LASSO against a dataset generated by a sparse linear combination of the 13 basis functions

$$\phi(x) = (x, \sin(0.4\pi x \cdot 1), \ldots, \sin(0.4\pi x \cdot 12)). \tag{17}$$

In comparison with ridge regression and vanilla linear regression, weights produced by LASSO were sparse (as expected, since the underlying generator was sparse)—most coefficients $w_j$ were driven to 0 as $\lambda$ increased; such was not the case with ridge regression, which we also tested. Consequently, LASSO fits generalized better, as is apparent from Figure 7.

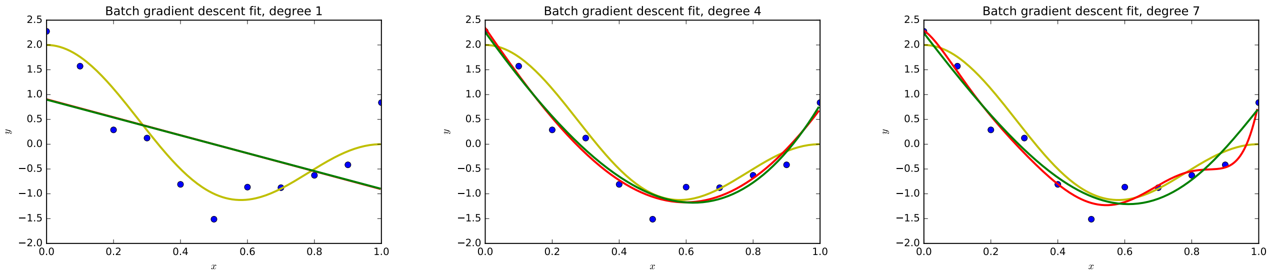

Figure 4: Polynomial fits of various degrees by batch gradient descent (green) of the dataset (blue) generated from a sum of cosines (yellow). The fits agree perfectly with the maximum-likelihood fits (red) at low degrees, but differ increasingly for higher degrees. Weights were initialized to zero.

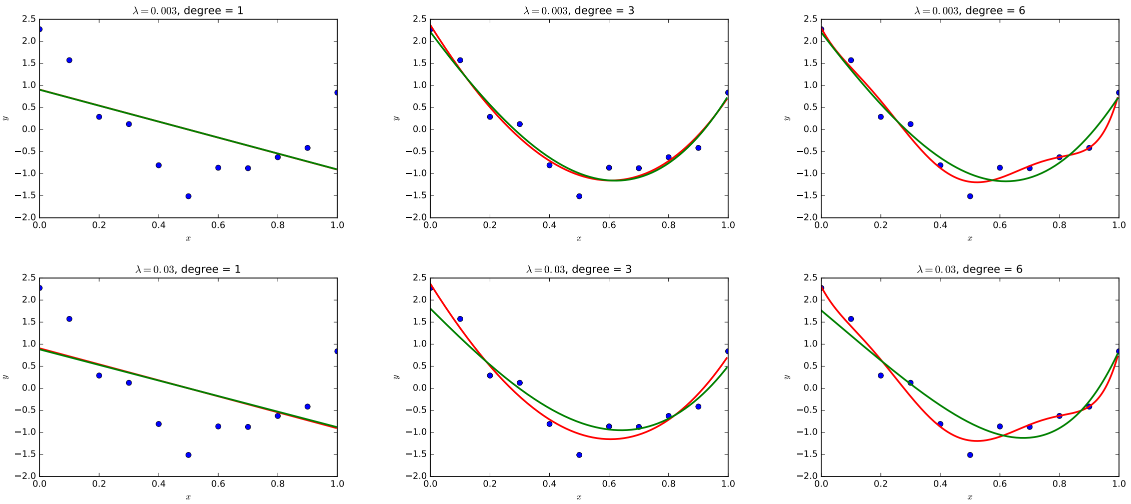

Figure 5: Regularized (green) and non-regularized (red) polynomial fits for the sum-of-cosines dataset.

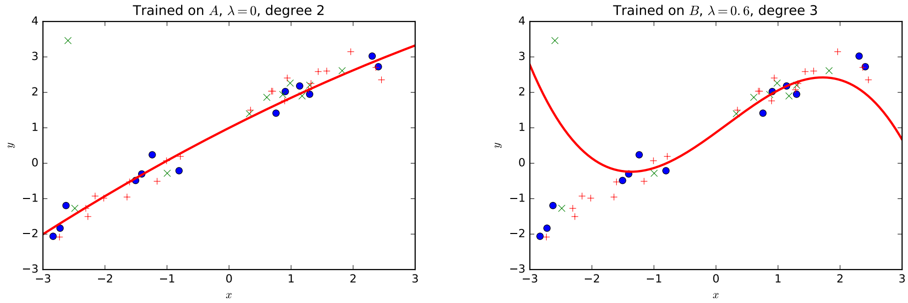

Figure 6: Best fit models for our dataset, where $A$ is plotted with blue dots, $B$ with green crosses, and $V$ with small plus signs. Note how the single outlier in $B$ skews the second fit significantly.

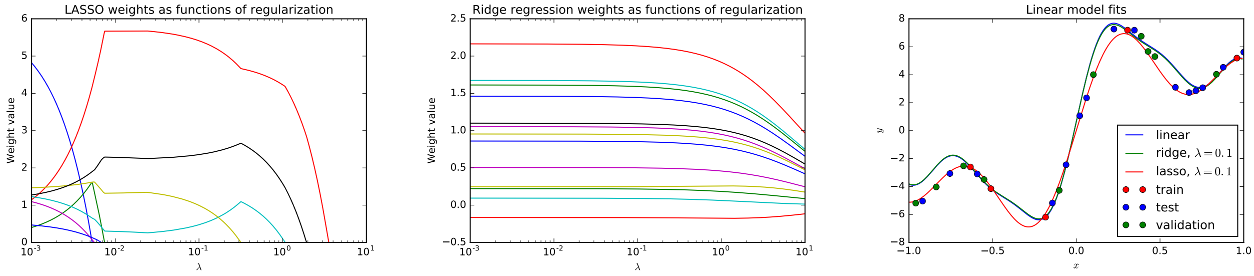

Figure 7: The weights produced by LASSO (left) and ridge regression (center) as their regularization parameters vary. Note how LASSO drives most weights to zero, and how linear changes to $\lambda$ result in linear changes to the weights (the curvature is an artifact of our horizontal log axis). (Right) Various linear-model fits of the training data. Note how LASSO generalizes much better to the test and validation datasets.