ENSC 813: Deep Learning Systems in Engineering

Kumar Abhishek

Assignment 1 June 4, 2021 Student ID: 301371793

Question 1

Generating synthetic data according to a p^{th} order polynomial

We use the following polynomial to generate N = 20 points.

$y = f(x) = 0.4x^3 - 2x^2 + 0.6x + 3$

We choose the range [-1.4, 5.4] to sample these points because the data distribution here is interesting: there is a crest and a trough followed by a huge peak.

We then calculate the labels for these sampled points using the polynomial above.

Next, in order to add random noise to the values, we sample random values from a zero mean unit variance Gaussian distribution and add it to the values. In Figure 1 below, we plot the sampled values as the blue crosses and the actual polynomial as the red curve. We can see that because of the noise added, the sampled points are close to the curve but not on it.

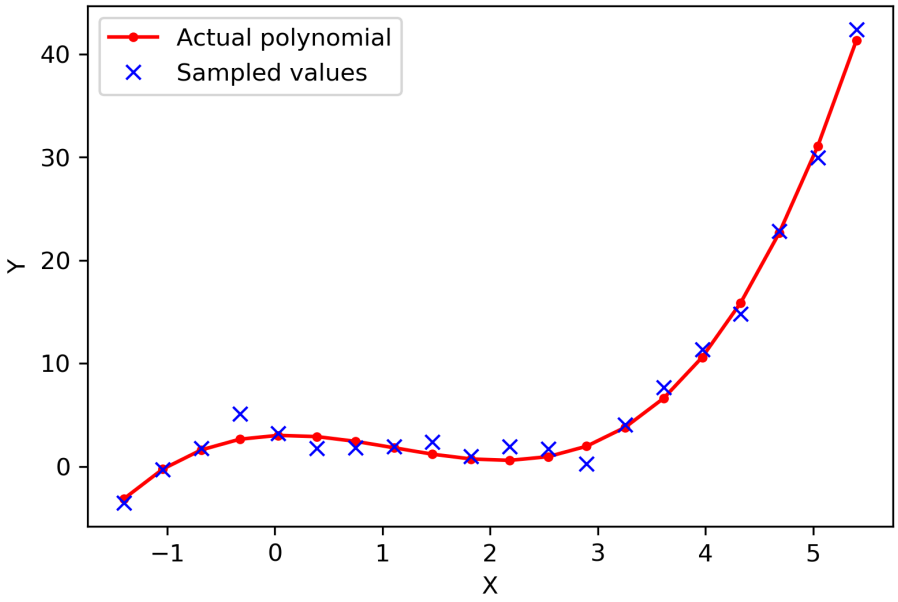

Figure 1: Generating synthetic data.

Fitting a two-parameter regression model to this data

We fit two-parameter regression model (i.e., a straight line) to this data. We do so by using the analytical method.

The parameters are calculated as [-0.71079588, 4.15566943]. Figure 2 shows the regression model overlaid on top of the data points.

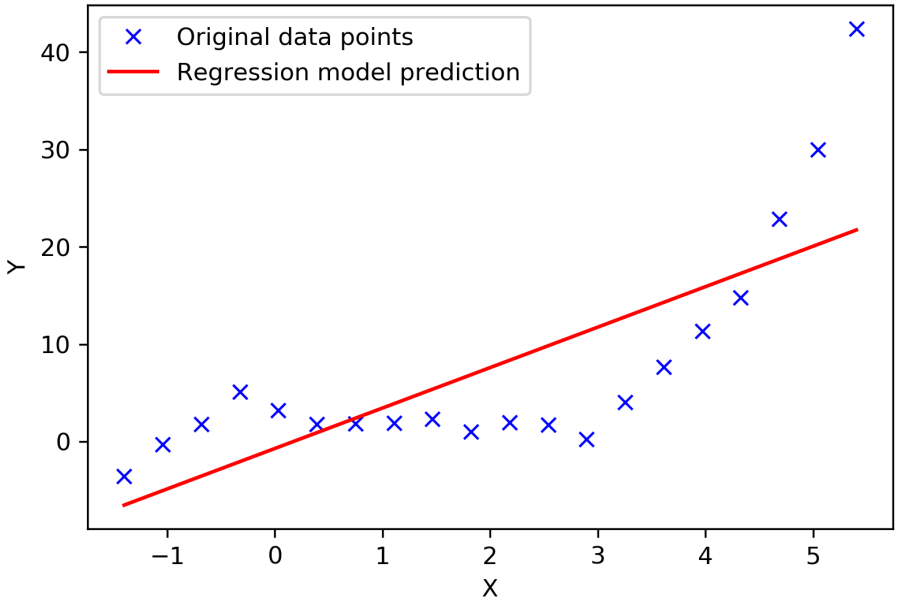

Figure 2: Two-parameter regression model fitted on the data.

Visualizing the cost surface of the regression model

We use the values of the parameters calculated above to guide our range of parameters for which the cost surface should be plotted. Therefore, we plot the cost surface of the model in the range of [-10, 10] for both the parameters.

We first construct a grid of points in the xy-plane based on the values of the two parameters in the range above, and then calculate the cost at each point in the grid. Finally, we plot the cost as a 3D surface. A colorbar is added to improve the readability of the surface plot. The cost surface plot is shown in Figure 3.

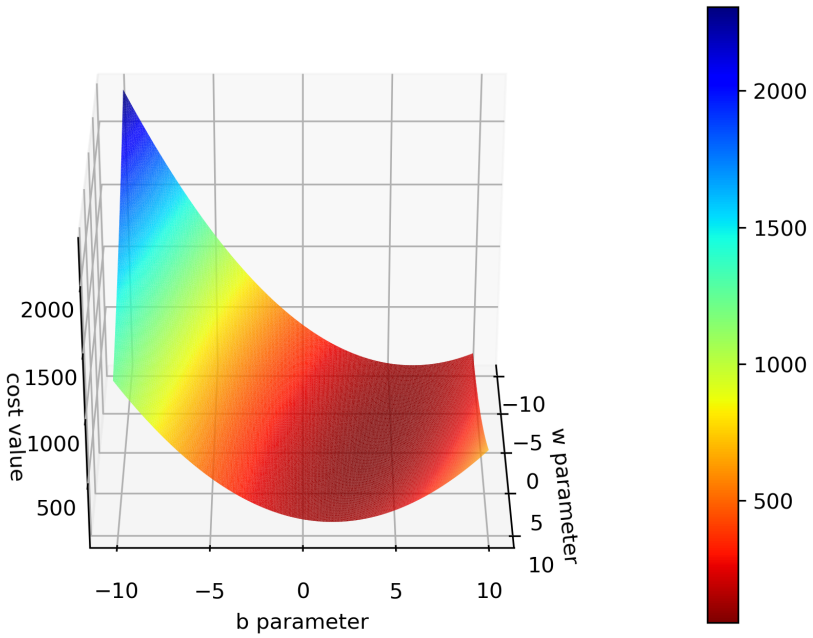

Figure 3: Surface plot of the model cost w.r.t. the two parameters.

Linear polynomial regression using analytical method

We perform linear polynomial regression on this data using the analytical method. We choose to fit a polynomial of order 3 to this data. However, the user can choose to fit a polynomial of any order by specifying it in the variable “fit_poly_order_ana”.

Figure 4 shows the learned model using the analytical method overlaid on top of the data points.

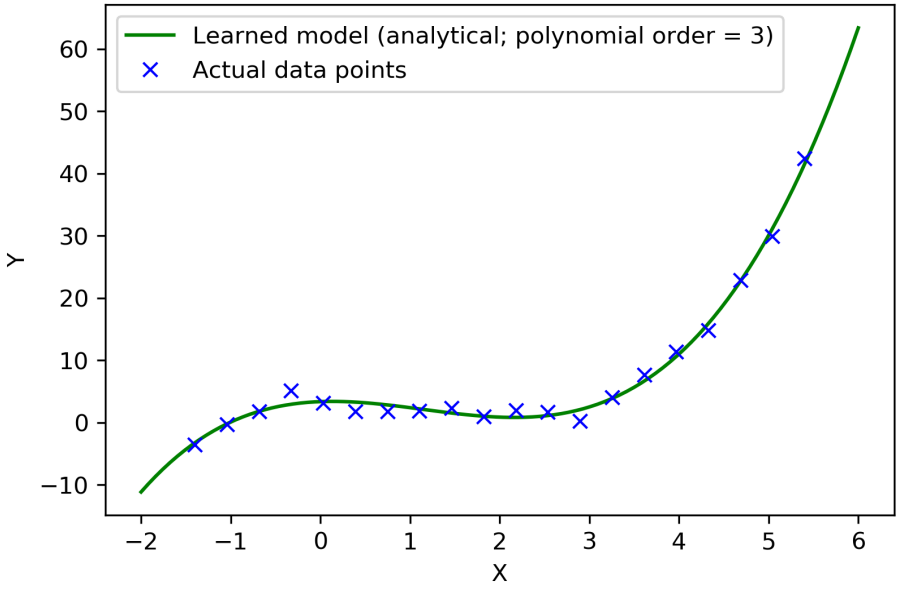

Figure 4: Linear polynomial regression using the analytical method.

Linear polynomial regression using gradient descent

We perform linear polynomial regression on this data using gradient descent. Similar to the analytical method, we choose to fit a polynomial of order 3 to this data. However, the user can choose to fit a polynomial of any order by specifying it in the variable “fit_poly_order_gd”.

Figure 5 shows the learned model using the analytical method overlaid on top of the data points.

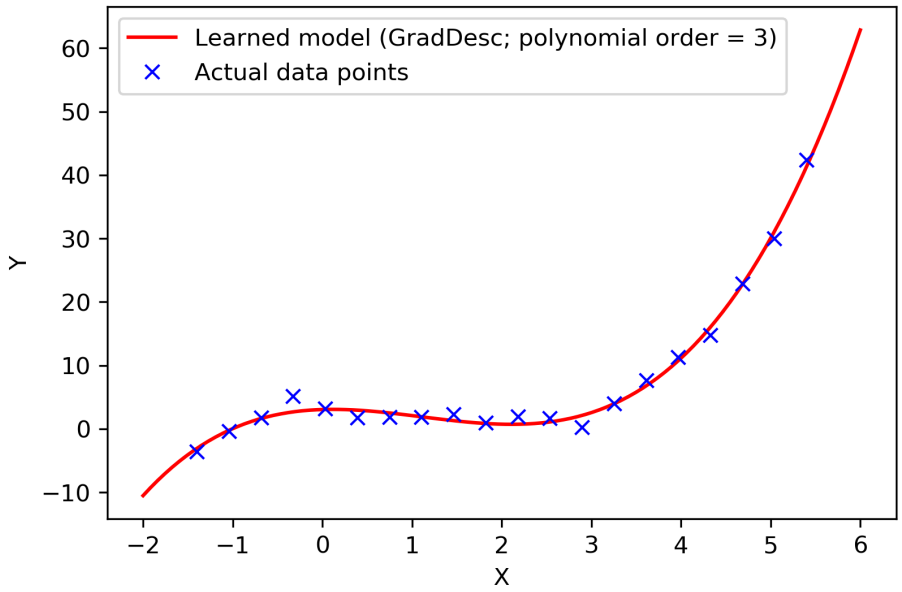

Figure 5: Linear polynomial regression using gradient descent.

Comparing the two models

We compare the results obtained using the two methods: analytical method and gradient descent. Below are the parameters obtained using the two methods. We see that the estimated parameters are very close to the actual ones, but not exactly the same, and this can be explained by the presence of noise in the data.

Actual parameters	**Estimated parameters (polynomial order = 3)**	
	Analytical method	**Gradient descent**
0.4	0.53634956	0.43088552
-2	-2.1246754	-1.98492865
0.6	0.61699189	0.59566546
3	3.37307521	3.05795251

Figure 6 shows the regression models learned by the two methods (both using a polynomial of order 3) overlaid on top of the data points. As we can see, the two models are almost exactly the same, except for a very minor difference near 0 on the x-axis.

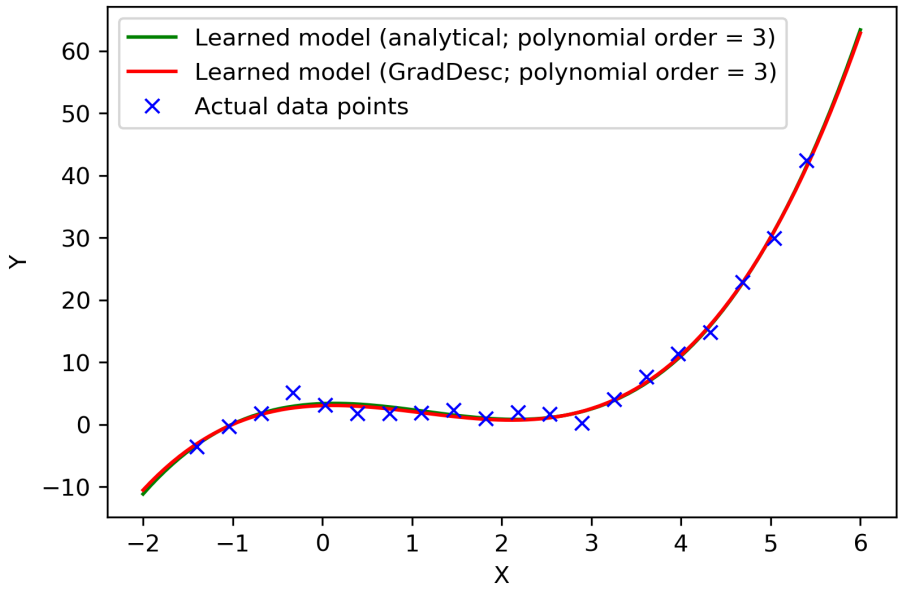

Figure 6: Comparing the regression models learned using the analytical method and gradient descent.

Studying the effect of learning rate in gradient descent

In order to explore the effect of varying the learning rate in gradient descent-based linear regression, we train polynomial regression models (polynomial order = 3) with 4 different learning rates: {1e-4, 5e-5, 1e-5, 1e-6}. Except the learning rate, all other hyperparameters for the models remain the same (polynomial order = 3, n_epochs = 1e5, batch_size = entire data).

Then, we plot the learned models for each of the 4 learning rates. Figure 7 shows the 4 trained models overlaid on top of the data points. We see that the model trained with the smallest learning rate (i.e., 1e-6; red curve) has the poorest fit to the data.

Next, we plot the cost values for each of these models as a function of the number of epochs in Figure 8. For improved readability, we plot the cost values (y-axis) on a logarithmic scale. Moreover, we also plot the cost values of the last 20k epochs to clearly show the difference in cost. We observe that there is an order of magnitude of difference between the cost values among these models. We conclude that models trained with (reasonably) larger learning rates converge faster than models trained with smaller learning rates.

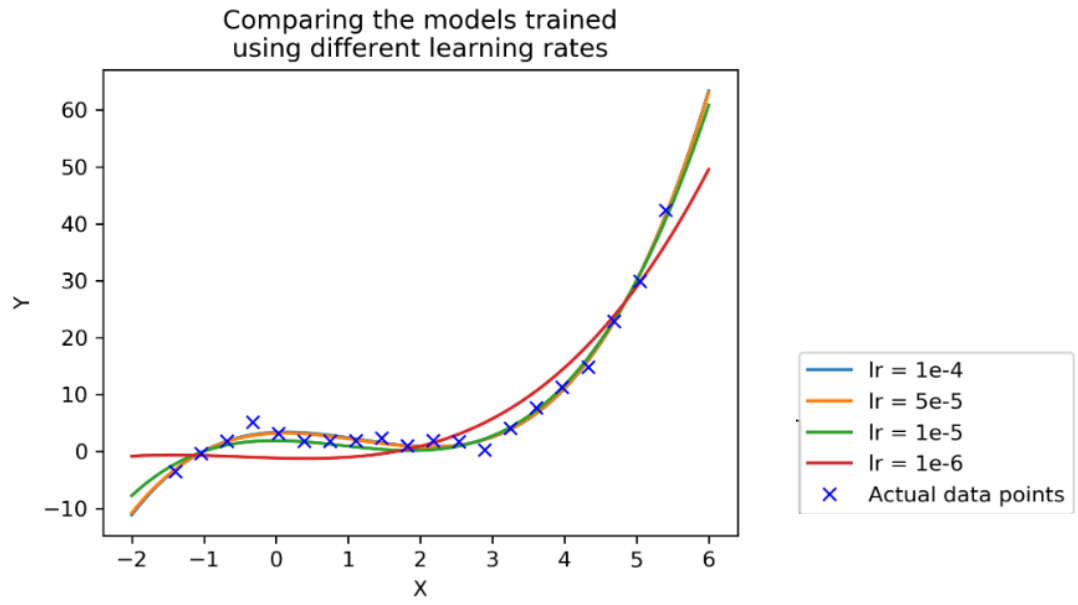

Figure 7: Comparing models trained with different learning rates.

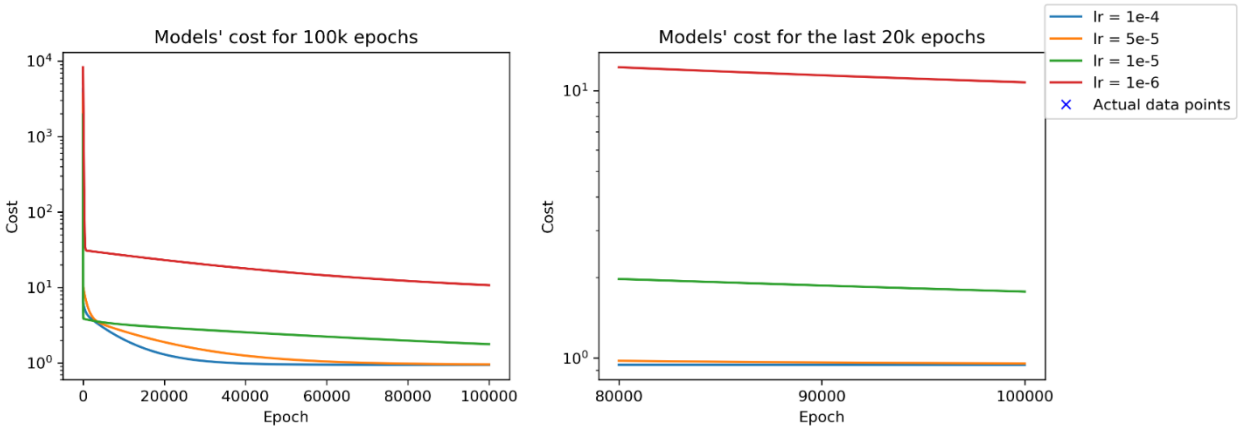

Figure 8: Comparing the costs for models trained with different learning rates.

Studying the effect of batch size in gradient descent

In order to explore the effect of varying the batch size in gradient descent-based linear regression, we train polynomial regression models (polynomial order = 3) with 4 different batch sizes: 1, 5, 10, and 20 (entire dataset) points. Except the batch size, all other hyperparameters for the models remain the same (polynomial order = 3, weight updates = 1e5, learning rate = 5e-5).

A point to be noted here is that when we perform stochastic (batch size = 1) or mini-batch gradient descent (1 < batch size < 20), the number of epochs is not the same as the number of

weight updates (also known as iterations). An epoch is defined as a single pass through the entire dataset, and so depending on the batch size, a single epoch can have many weight updates. Therefore, in order to study the effect of batch size in gradient descent, we compare models trained with different batch sizes but for the same number of weight updates (or iterations).

Figure 9 shows the plot of the models' costs for the 4 batch sizes in our experiments. Observe that stochastic gradient descent (batch size = 1; blue plot) looks like an envelope of a wave because of the large amount of oscillations between consecutive cost values. This is because with stochastic gradient descent, only one training sample is used at a time to update the model's parameters, and therefore, this can lead to large variations in consecutive parameters, and consequently in the model's cost.

As the batch size increases (batch sizes of 5 and 10; green and red plots respectively), this variation between consecutive weight updates becomes smaller because a larger number of data points is used to update the weights, and therefore consecutive cost values exhibit lesser oscillation.

Finally, when using all the data at a time to update the model's parameters (orange plot), there are no abrupt weight updates and the model converges (almost) monotonically.

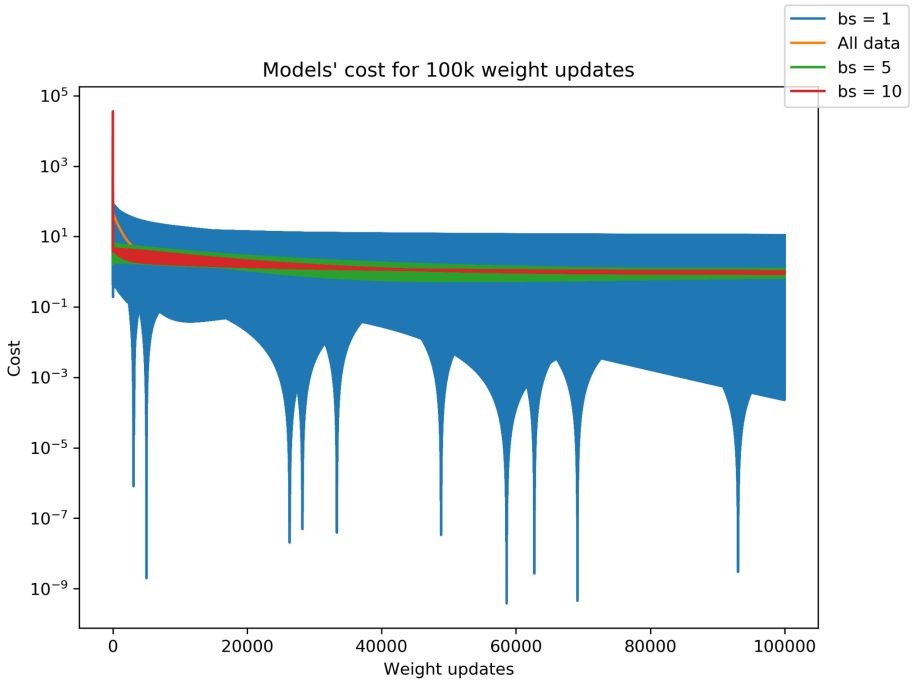

Figure 9: Visualizing the models' costs when trained with different batch sizes.

Plotting model parameters for each iteration

In order to visualize how each model parameter changes during the course of gradient descent, we perform a linear polynomial regression over the data points and plot each parameter's value w.r.t. epochs trained. We choose to fit a polynomial of order 3, and therefore we have 4 parameters to update and keep track of.

Figure 10 shows the iterative updates of the model parameter values. Notice how they converge towards the respective ground truth values: "w_0" (blue) approaches 0.4, "w_1" (orange) approaches -2, "w_2" (green) approaches 0.6, and "b" (red) approaches 3.

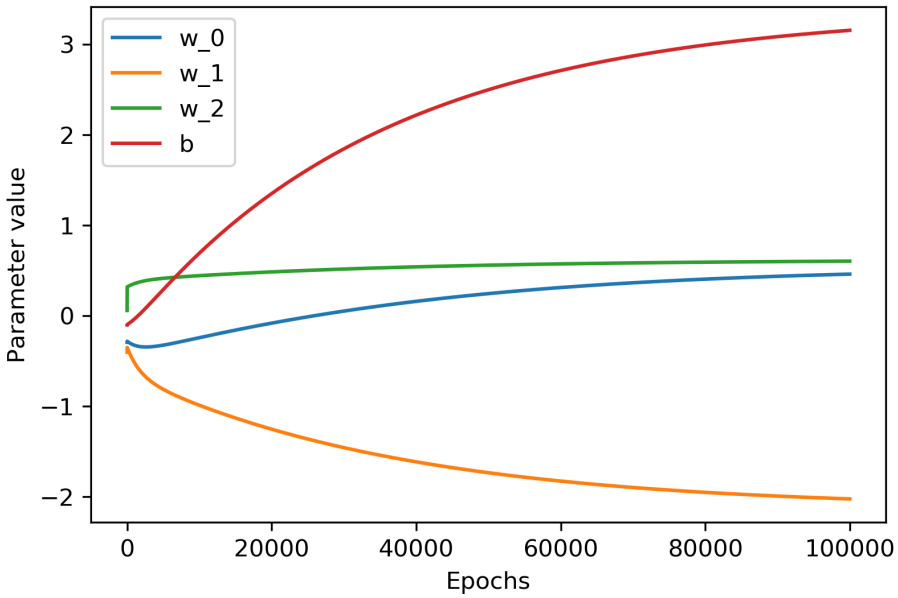

Figure 10: Visualizing the values of model parameters w.r.t. epochs trained.

Plotting model parameters as a curve on the cost surface

In order to visualize how the model parameters converge, we plot their values over the course of training on top of the model's cost surface. Since we can only visualize in 3D, we use a two-parameter regression model with parameters "w" and "b", which are plotted in the x-y plane, with the model's cost on the z-axis. This is therefore the same cost surface as plotted above.

Next, we train a two-parameter regression model on the data for 1e6 epochs and keep track of the parameters and the model cost at each epoch. Once the model is trained, we plot each epoch's cost as a function of the respective epoch's "w" and "b" as a curve (black plot) on top of the model's cost surface, as shown in Figure 11.

Looking at the black curve, we observe that from the time of initialization (epoch 0), as the model parameters ("w" and "b") are updated using gradient descent, their cost keeps

decreasing, and therefore the black curve moves accordingly. The black curve looks small compared to the cost surface because the surface has been plotted over the range [-10, 10] for both the parameters, whereas the parameters observed during the model training are limited to a much smaller range.

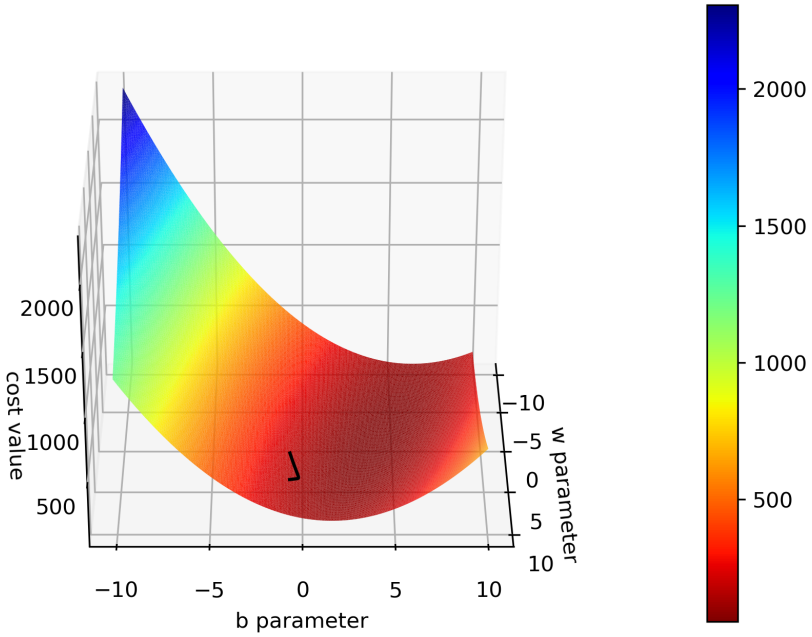

Figure 11: Model parameter convergence on the cost surface.

Studying the effect of number of training samples in gradient descent

In order to explore the effect of varying the number of training samples in gradient descent-based linear regression, we train multiple regression models with different number of training samples: 5, 10, 25, and 50 points. Moreover, to study the impact of polynomial order, we fit polynomials of order 1, 2, and 3. All other hyperparameters for the models remain the same (weight updates = 1e5, learning rate = 5e-5).

Figure 12 shows the trained models with different dataset sizes when using polynomials of order 1 (Figure 12 (a)), 2 (Figure 12 (b)), and 3 (Figure 12 (c)).

The difference is most conspicuous when fitting a 1st order polynomial (Figure 12 (a)), where we can see that despite being a two-parameter model, the model with 50 points (red curve) captures the distribution quite well, taking into account the outliers after x = 4, whereas the model with 5 points (blue curve) does not take into the outliers into account.

Similarly, with a 2nd order polynomial (Figure 12 (b)), the model with 50 points (red curve) is the parabola that fits all the points the best, including the outliers; and the opposite effect is observed with the model with 5 points (blue curve) which ignores (most) outliers.

Finally, with a 3rd order polynomial (Figure 12 (c)), we observe a similar trend but it is much less perceptible. We can see that the model with 50 points (red curve) fits the data points the best, while the model with 5 points (blue curve) misses many points and performs the worst.

Therefore, to conclude, the number of training samples plays an important role in determining how accurately the model fits to the data, and the general trend observed here is that the larger the number of training samples, the better the model learns to represent the data distribution.

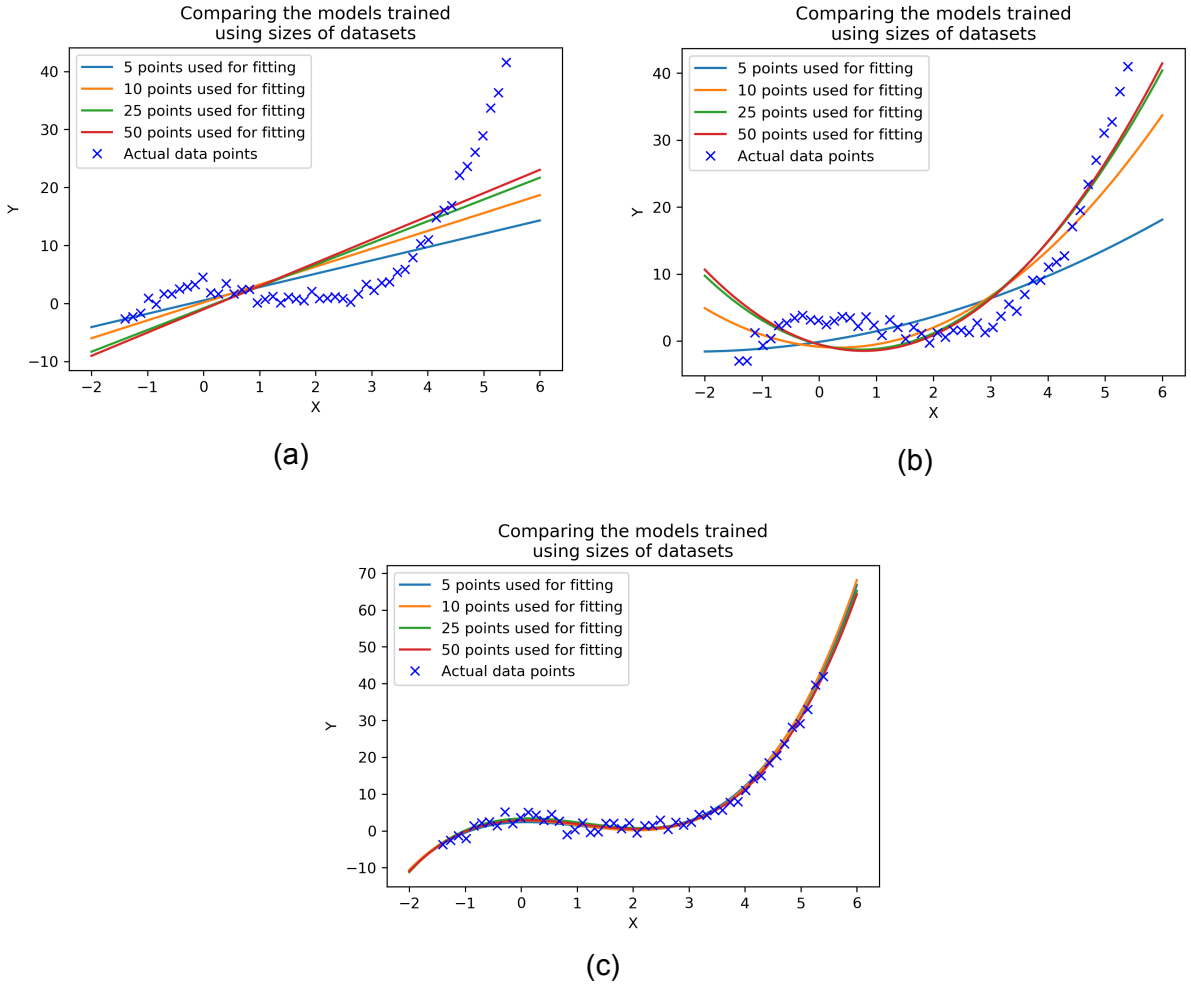

Figure 12: Studying the effect of training dataset size in gradient descent when fitting polynomials of order 1, 2, and 3.

Repeating the experiments above for data sampled from a 2nd order polynomial

We use the following 2nd order polynomial to generate N = 20 points.

$y = f(x) = x^2 - 3x + 1$

We choose the same range as before, i.e., [-1.4, 5.4], to sample these N points. As before, we calculate the labels for these points based on the 2nd order polynomial and then corrupt the labels by adding Gaussian noise to the values. In Figure 13 below, we plot the sampled values as the blue crosses and the actual polynomial as the red curve. We can see that because of the noise added, the sampled points are close to the curve but not on it.

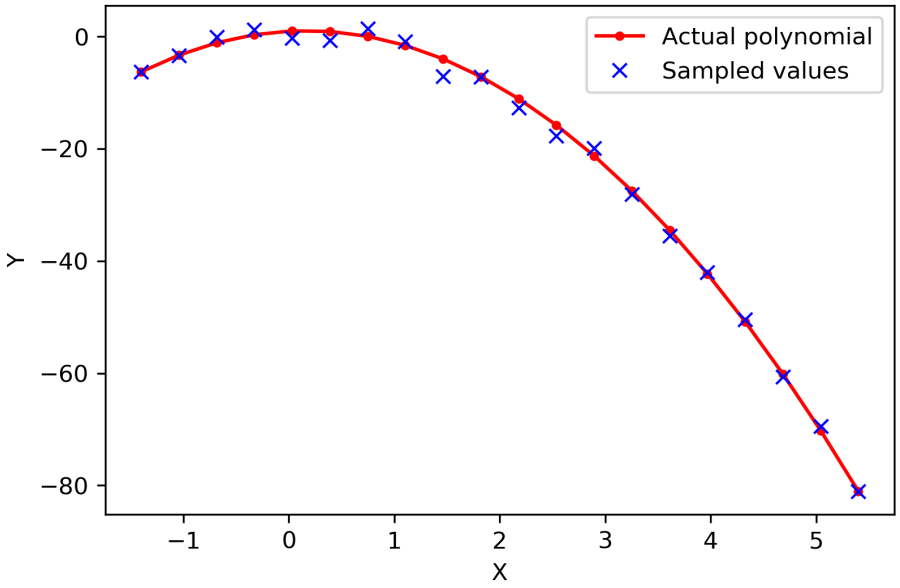

Figure 13: Generating synthetic data from a 2nd order polynomial.

Fitting regression models to the data using analytical method and gradient descent

We first fit a 2nd order polynomial to this data using the analytical method. After that, we use gradient descent to fit a 1st, a 2nd, and a 3rd order polynomial to this data. For all the gradient descent methods, the hyperparameters remain the same (learning rate = 5e-5, n_epochs = 1e5).

Figure 14 shows these 4 estimated regression models overlaid on top of the data points. As we can see, the analytical model and the gradient descent models trained with polynomials of order 1 and 2 fit the data quite well and are difficult to distinguish in the plot (all of them are overlaid

on top of each other in the black curve). Interestingly however, the regression model trained with a polynomial of order 3 (blue curve) yields a poor fit and this can be attributed to improper choice of hyperparameters. This is because in order to ensure a fair comparison of gradient descent with polynomials of different order, we have stuck to the same set of hyperparameters for all the gradient descent models.

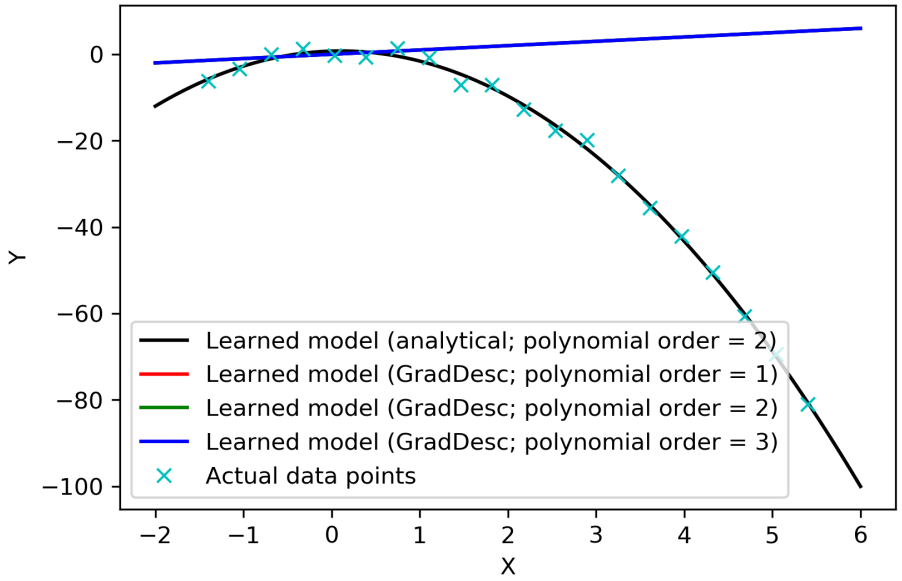

Figure 14: Comparing models trained with analytical method and gradient descent on data generated according to a 2nd order polynomial.

Repeating the experiments above for data sampled from a 1st order polynomial

We use the following 1st order polynomial to generate N = 20 points.

$y = f(x) = 0.5x + 0.7$

We choose the same range as before, i.e., [-1.4, 5.4], to sample these N points. As before, we calculate the labels for these points based on the 2nd order polynomial and then corrupt the labels by adding Gaussian noise to the values. In Figure 15 below, we plot the sampled values as the blue crosses and the actual polynomial as the red curve. We can see that because of the noise added, the sampled points are close to the curve but not on it.

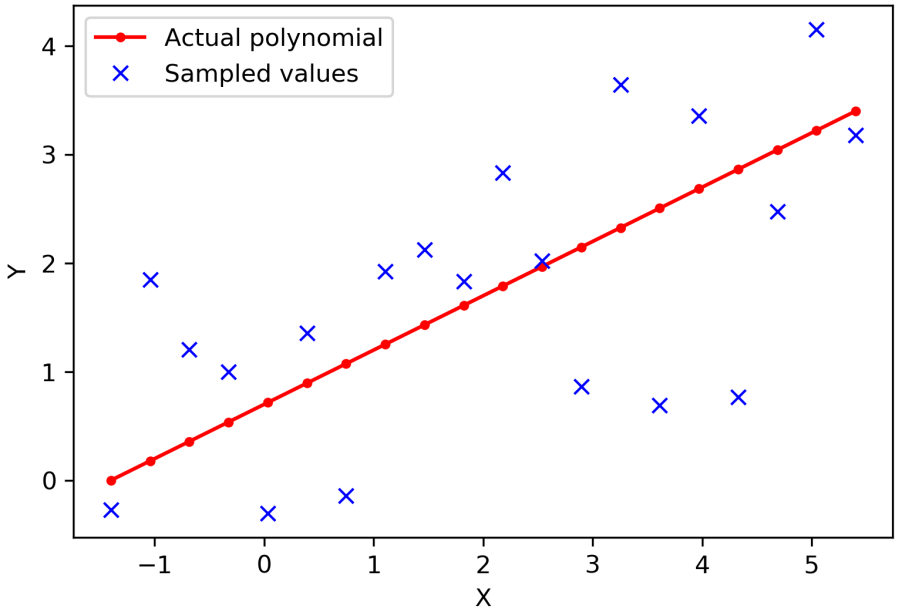

Figure 15: Generating synthetic data from a 1st order polynomial.

Fitting regression models to the data using analytical method and gradient descent

We first fit a 1st order polynomial to this data using the analytical method. After that, we use gradient descent to fit a 1st, a 2nd, and a 3rd order polynomial to this data. For all the gradient descent methods, the hyperparameters remain the same (learning rate = 5e-5, n_epochs = 1e5).

Figure 16 shows these 4 estimated regression models overlaid on top of the data points. As we can see, the analytical model and the gradient descent models trained with polynomials of order 1 and 2 fit the data quite well and are difficult to distinguish in the plot (all of them are overlaid on top of each other in the black curve). The regression model trained with a polynomial of order 3 (blue curve) yields a slightly worse fit and is more sensitive to outliers. Again, this can be attributed to improper choice of hyperparameters because in order to ensure a fair comparison of gradient descent with polynomials of different order, we have stuck to the same set of hyperparameters for all the gradient descent models.

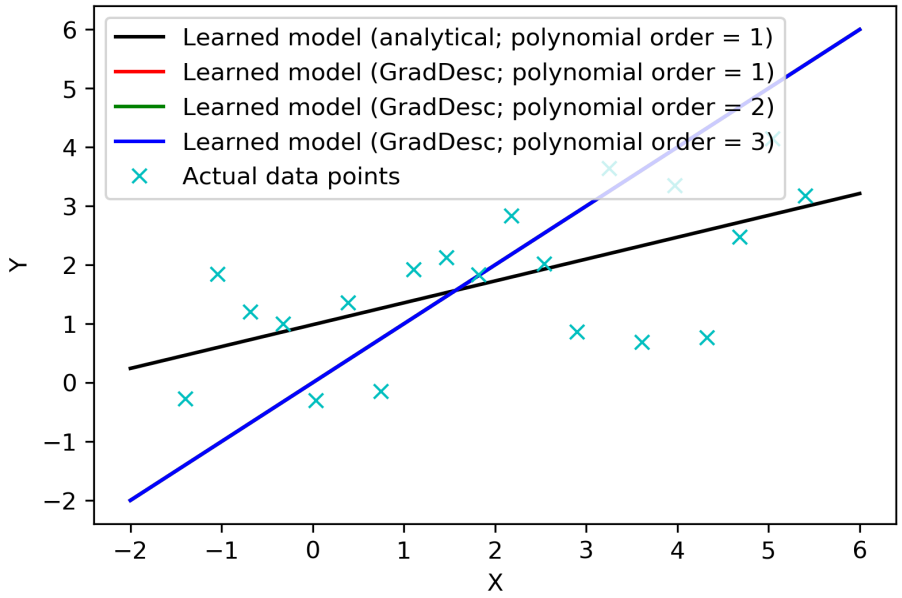

Figure 16: Comparing models trained with analytical method and gradient descent on data generated according to a 2nd order polynomial.

Question 2

Reading and plotting the data

We first understand the dataset by visualizing the distribution of weight versus height among the subjects.

Figure 17 shows a scatter plot of the weight versus height distribution, where each blue dot represents a subject.

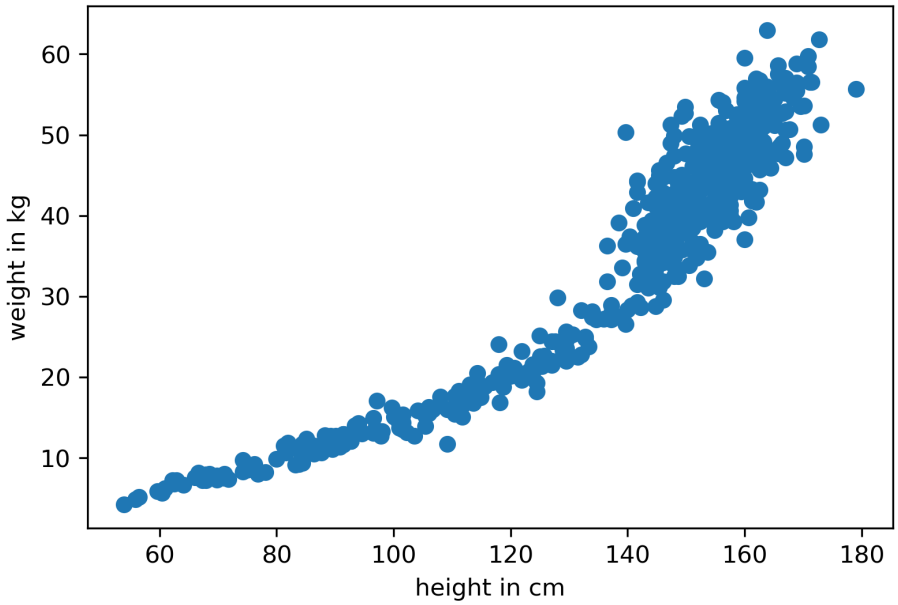

Figure 17: A scatter plot of the height and the weight of the subjects in the dataset.

Fitting a two-parameter regression model to predict weight from height

In order to predict the weight from a given height, we perform linear regression to fit a two-parameter model (a straight line) to this data.

The model's parameters are calculated as [0.5016993362336575, -33.75613690832983]. The learned model (red) is overlaid on top of the data points (blue) in Figure 18 and we can see that it captures the relationship between weight and height quite well, while also taking the outliers into account.

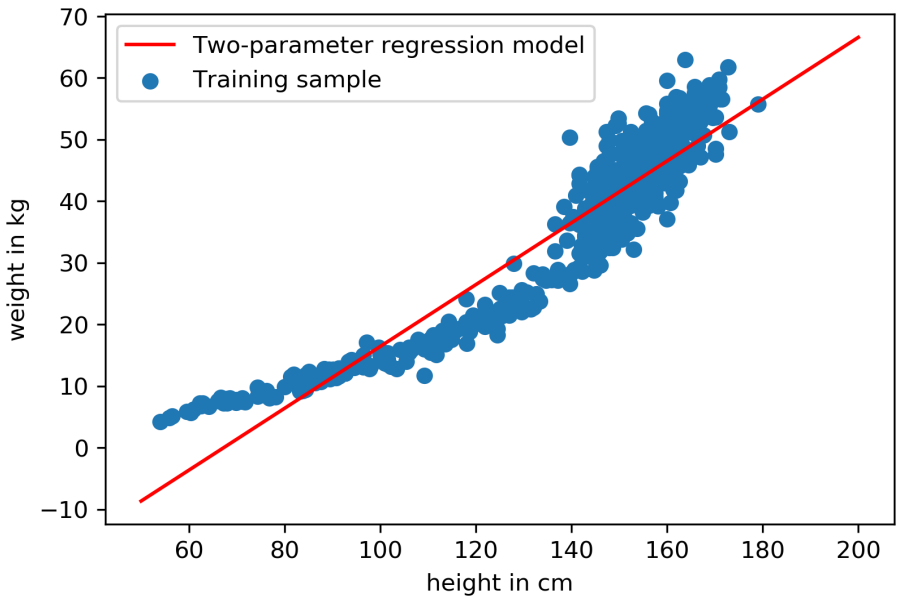

Figure 18: Two-parameter regression model to predict weight from height.

Visualizing the cost surface of the regression model

We use the values of the parameters calculated above to guide our range of parameters for which the cost surface should be plotted. Because there is a very large difference in the values of the two parameters and plotting 3D surfaces is computationally expensive, we plot the cost surface of the model in the range of [-5, 5] and [-38, -28] for the two parameters respectively. This ensures that we limit our cost surface plot to a region around the global minimum.

We first construct a grid of points in the xy-plane based on the values of the two parameters in the range above, and then calculate the cost at each point in the grid. Finally, we plot the cost as a 3D surface. A colorbar is added to improve the readability of the surface plot. The cost surface plot is shown in Figure 19.

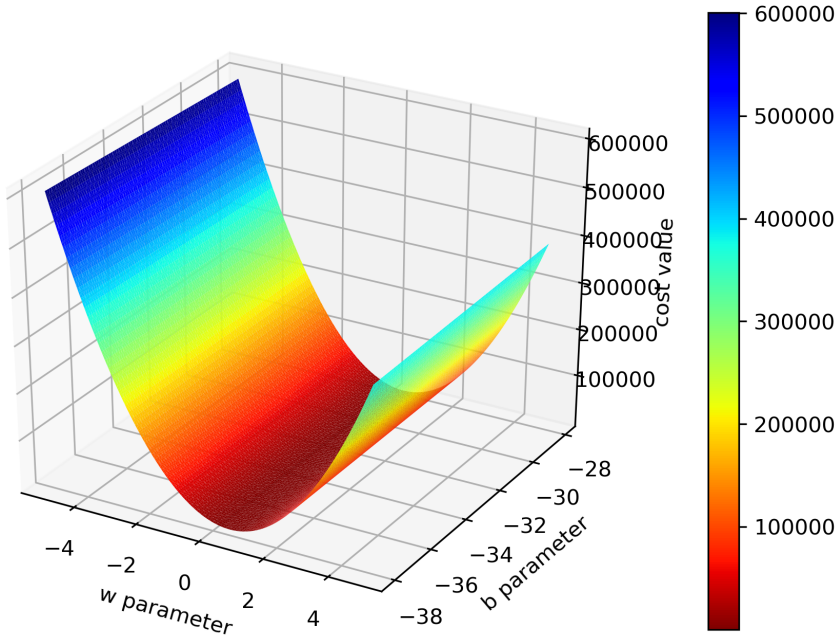

Figure 19: Surface plot of the model cost w.r.t. the two parameters.

Predicting weight from height using analytical method

We use the analytical method to fit a two-parameter model (a straight line) for predicting the weight of a subject given the height.

The model's parameters are calculated as [0.48125474, 21.48848913]. The learned model (red) is overlaid on top of the data points (blue) in Figure 20 and we can see that it captures the relationship between weight and height quite well, while also taking into account the outliers and the non-linear distribution of the scatter plot.

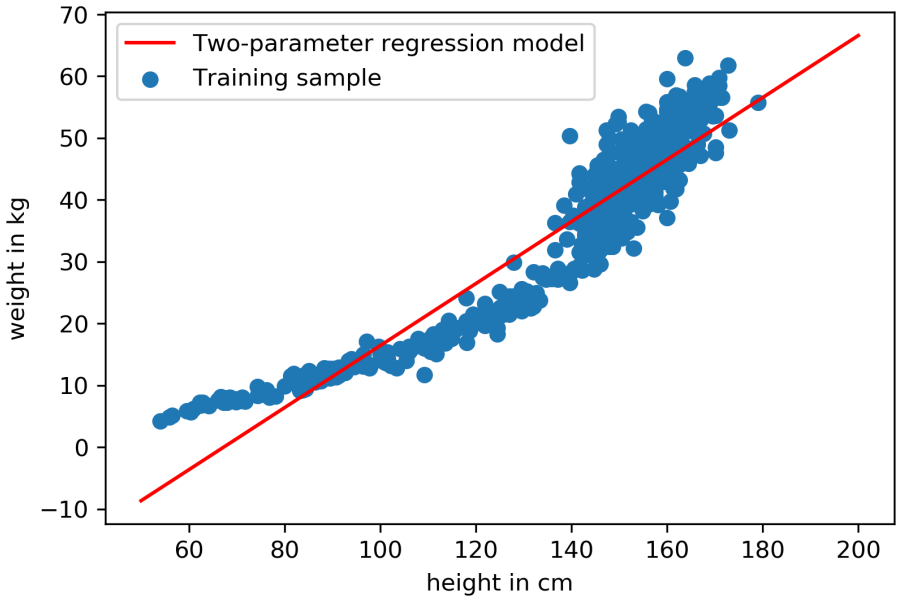

Figure 20: Two-parameter regression model to predict weight from age.

Predicting weight from age using analytical method

We use the analytical method to fit a two-parameter model (a straight line) for predicting the weight of a subject given the age.

The model's parameters are calculated as [0.48125474, 21.48848913]. The learned model (red) is overlaid on top of the data points (blue) in Figure 21 and we can see that it captures the relationship between weight and age quite well, despite the non-linear distribution of the scatter plot.

Figure 21: Two-parameter regression model to predict weight from age.

Multivariate linear regression to predict weight from all of height, age, and sex

We use the analytical method to fit a multivariate linear regression model for predicting the weight of a subject given the height, the age, and the sex.

Since there are 3 features to predict from, this model is a 3rd order polynomial with 4 parameters (i.e., 4 polynomial coefficients). The estimated parameters for [height, age, sex, b] are [0.4734134, 0.05050421, 0.92548938, -31.76446244].