Machine Learning: Assignment 1, Part 1

Pedro Pereira Sarmento

Queen Mary, University of London

November 15, 2019

1 Linear Regression with One Variable

The function presented below calculates the hypothesis for a given X, θ (theta) and training example (i). In this case, θ contains one bias term and one parameter.

```
def calculate_hypothesis(X, theta, i):
    hypothesis = X[i,0] * theta[0] + X[i,1] * theta[1]
    return hypothesis
```

Next, the same function is used in our updated gradient descent function, presented below, for both θ_0 and θ_1.

```
# update temporary variable for theta_0
sigma = 0.0
for i in range(m):
    hypothesis = calculate_hypothesis(X, theta, i)
    output = y[i]
    sigma = sigma + (hypothesis - output)
theta_0 = theta_0 - (alpha/m) * sigma
# update temporary variable for theta_1
sigma = 0.0
for i in range(m):
    hypothesis = calculate_hypothesis(X, theta, i)
    output = y[i]
    sigma = sigma + (hypothesis - output) * X[i, 1]
theta_1 = theta_1 - (alpha/m) * sigma
```

Trying different values of the learning rate α produced changes regarding the cost function and the time it takes to find an optimum value. Using a very high learning rate ($\alpha = 10$) renders the process unstable, overshooting updates to θ and missing the optima of the cost function, as observable in Figures 1 and 2.

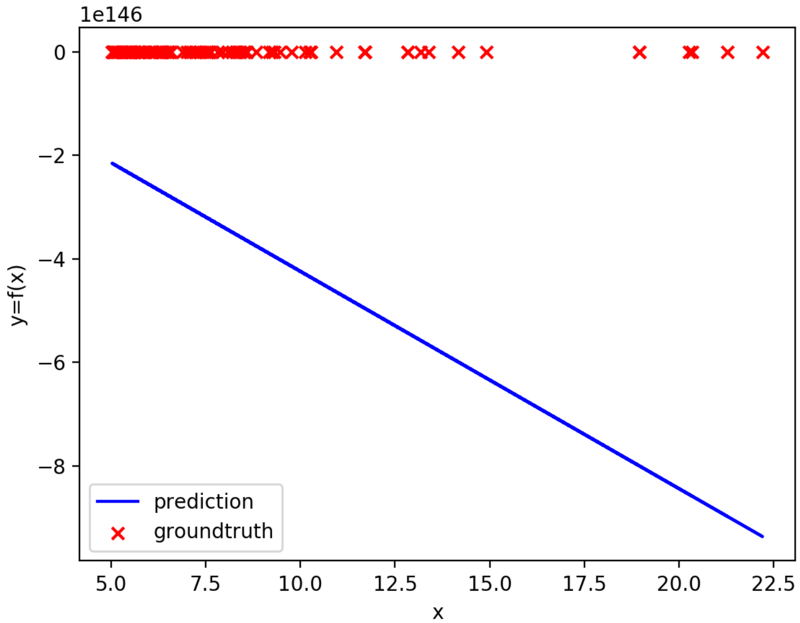

Fig. 1: Hypothesis function for $\alpha = 10$.

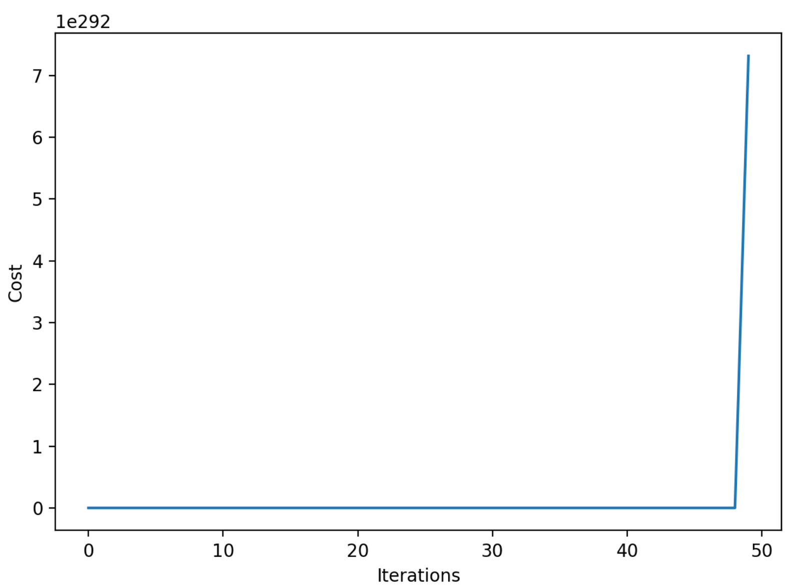

Fig. 2: Cost function by iteration, for $\alpha = 10$.

For $\alpha = 1$ and $\alpha = 0.1$, although a noticeable reduction in the cost function, the process remains unstable, presenting an inadequate prediction.
By considering a learning rate $\alpha = 0.01$ it was found that the cost function gets close to its optima, as presented in Figures 3 and 4.
When choosing a learning rate of $\alpha = 0.02$, the cost function gets closer to its global optima much faster than any other previous attempt.s Increasing α to 0.03 renders the process once again unstable. This suggest that a compromise should be met concerning α: a very high learning rate causes the parameters to change too quickly and ultimately leads the process to failure, and a very slow learning rate will take too many iterations to find a model of best fit, meaning we would need a substantial increase in the number of iterations before the global optima for a cost function is reached (e.g. in our case, the default 50 iterations aren't sufficient for a learning rate α of 0.001).

2 Linear Regression with Multiple Variables

The function shown bellow generalises our previous hypothesis function for a specified training example, into a new one that performs the same procedure for any number of extra variables.

```
def calculate_hypothesis(X, theta, i):
    hypothesis = 0.0
    numVar = X.shape[1]
    for var in range(numVar):
        hypothesis += X[i,var] * theta[var]
    return hypothesis
```

Similarly, gradient descent has been revised such that it performs optimization for any number of theta parameters.

```
for it in range(iterations):
        theta_temp = theta.copy()
        sigma = np.zeros((len(theta)))
        for i in range(m):
            hypothesis = calculate_hypothesis(X,theta,i)
            output = y[i]
            for k in range(len(theta_temp)):
                sigma[k] = sigma[k] + (hypothesis - output) * X[i,k]
        for k in range(len(theta_temp)):
            theta_temp[k] = theta_temp[k] - (alpha/m) * sigma[k]
```

With a learning rate $\alpha = 0.1$, we get $\theta = [3.3865\times10^5, 1.0332\times10^5, -0.0047\times10^5]$ and the cost function converges after approximately 20 iterations (Figure 16).

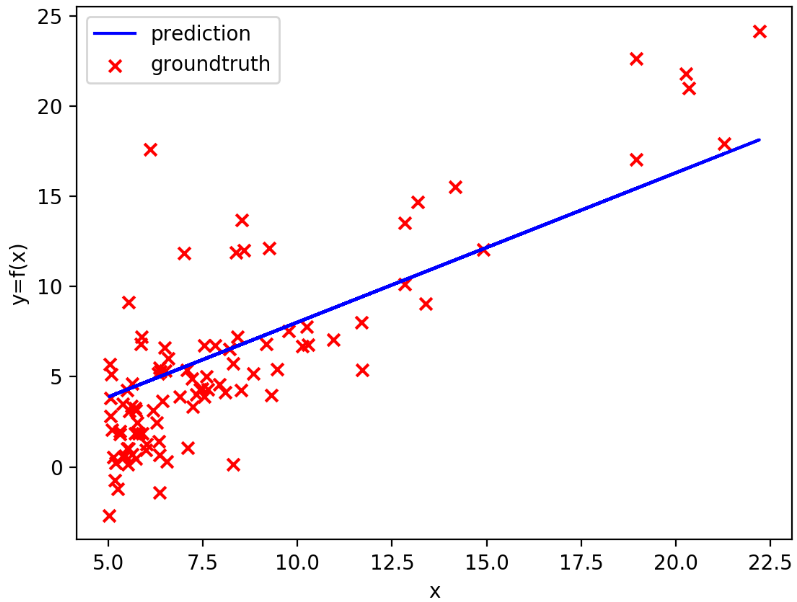

Fig. 3: Hypothesis function for $\alpha = 0.01$.

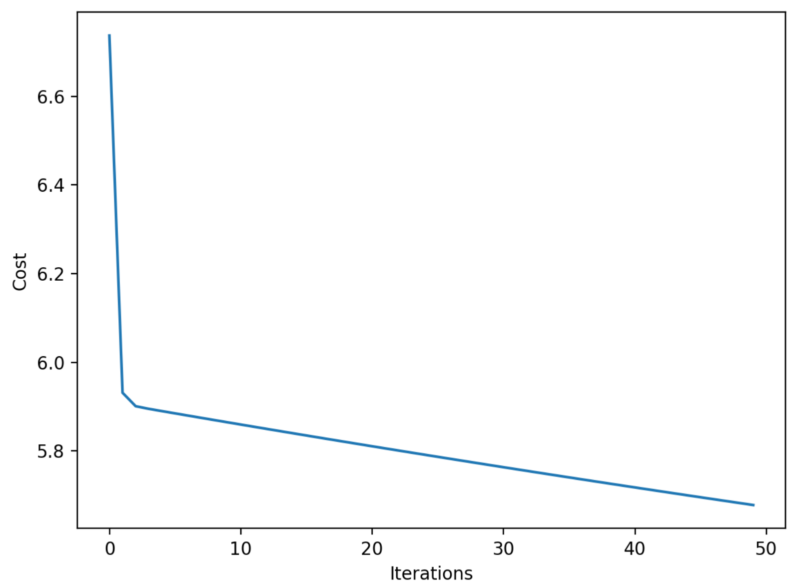

Fig. 4: Cost function by iteration, for $\alpha = 0.01$.

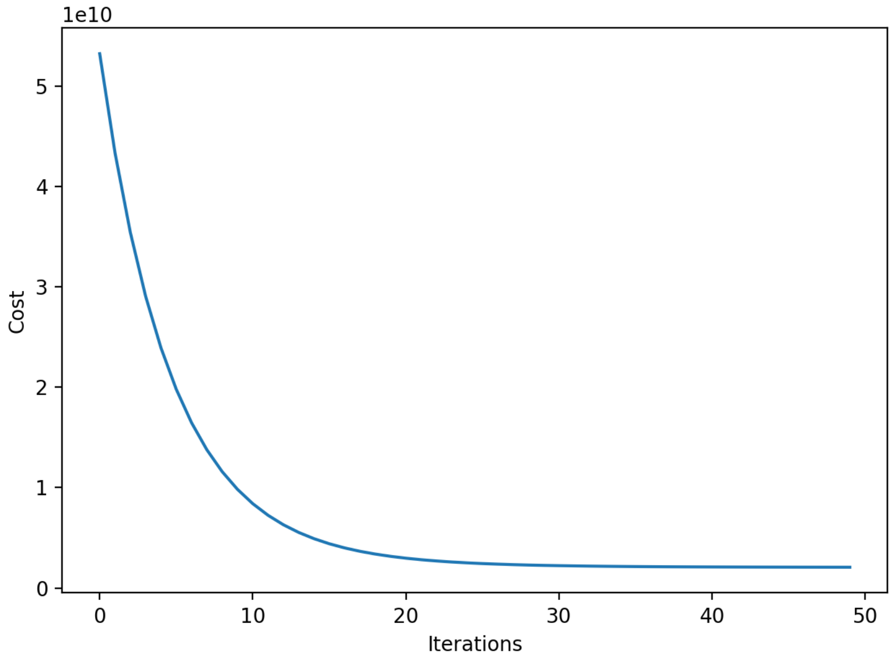

Fig. 5: Cost function by iteration,for $\alpha = 0.1$.

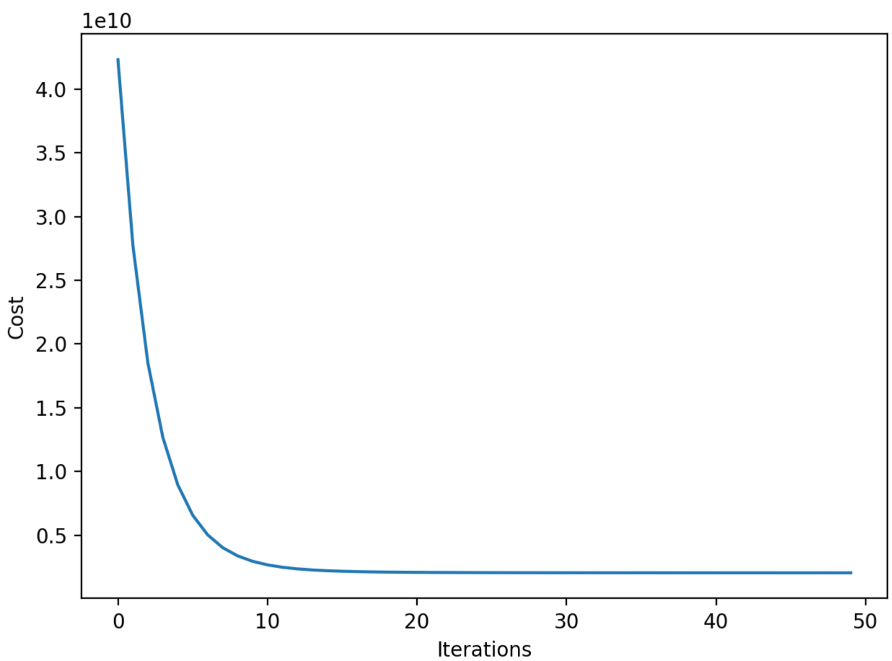

Fig. 6: Cost function by iteration,for $\alpha = 0.2$.

By setting $\alpha = 0.2$, we obtain $\theta = [3.4040 \times 10^5, 1.0886 \times 10^5, -0.0599 \times 10^5]$ and the cost function tends towards its minimum faster, as presented in Figure 17. Trying different learning rate values led to the conclusion that $\alpha = 0.5$ rapidly reaches the minimum, taking only five iterations to do so. In this scenario, shown in Figure 7, $\theta = [3.4041 \times 10^5, 1.0944 \times 10^5, -0.0658 \times 10^5]$, highlighting the importance of the size in square feet parameter when compared to the number of bedrooms.

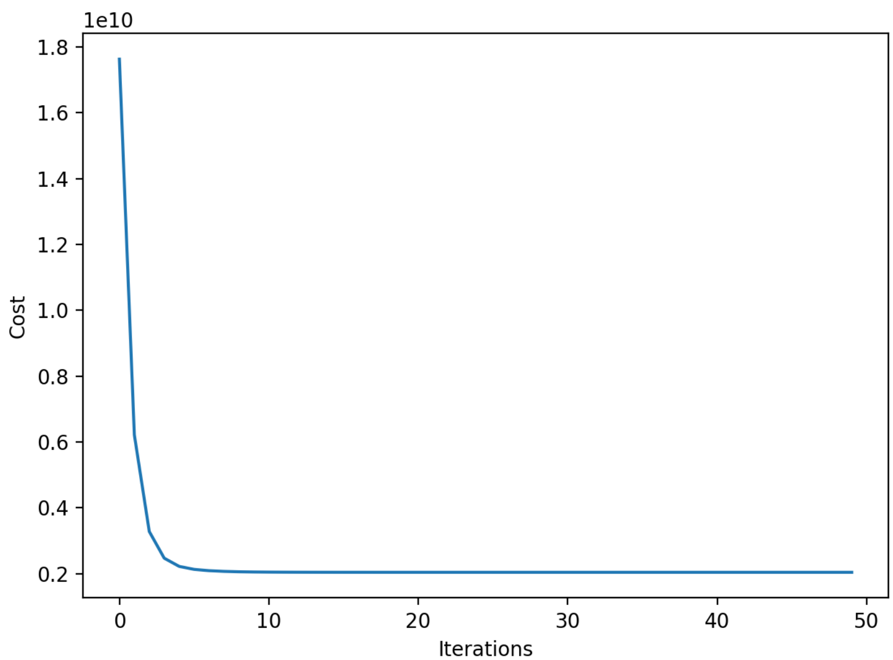

Fig. 7: Cost function by iteration,for $\alpha = 0.5$.

The code for testing the trained hypothesis for houses prices of a 1650 sq. ft, 3 bedroom house and a 3000 sq. ft, 4 bedroom house is shown below.

```
# Normalized feature vectorss
test = np.array([[(1650-mean_vec[0,0])/std_vec[0,0],
                   (3-mean_vec[0,1])/std_vec[0,1]],
                 [(3000-mean_vec[0,0])/std_vec[0,0],
                  (4-mean_vec[0,1])/std_vec[0,1]]])
# Bias
ones = np.array([[1], [1]])
```

```
# Normalized feature vectors with biases
test_f = np.append(ones, test, axis =1)
# Display predicted prices for both examples
print(calculate_hypothesis(test_f, theta_final, 0))
print(calculate_hypothesis(test_f, theta_final, 1))
```

The results for the price of both house examples are 2.9308×10^5 and 4.7228×10^5 respectively.

3 Regularized Linear Regression

Below is presented the updated gradient descent, using the *compute_regularised_cost* function instead of *compute_cost*.

```
# Append current iteration's cost to cost_vector
        iteration_cost = compute_cost_regularised(X, y, theta,l)
        cost_vector = np.append(cost_vector, iteration_cost)
```

Also, in *gradient_descent*, λ (l), the regularisation term, is applied to θ (theta) on each update iteration, excluding the bias term.

```
for k in range(len(theta_temp)):
            if k == 1:
                # Bias term
                theta_temp[k] = theta_temp[k] - (alpha/m) * sigma[k]
            else:
                # Apply regulariser to all other parameters
                theta_temp[k] = theta_temp[k] * (1- (alpha * (l/m)))
                - ((alpha * 1)/ m) * sigma[k]
```

After trying multiple values for α (including 10, 5, 2, 1, 0.5, 0.2, 0.1, 0.05, 0.01) it was found that $\alpha = 1.4$ generated the best optimization (Minimum Cost = 0.00749) on iteration number 200 (for a maximum of 200 iterations).
In the following figures is presented the effect of increasing λ on the cost function and the hypothesis.

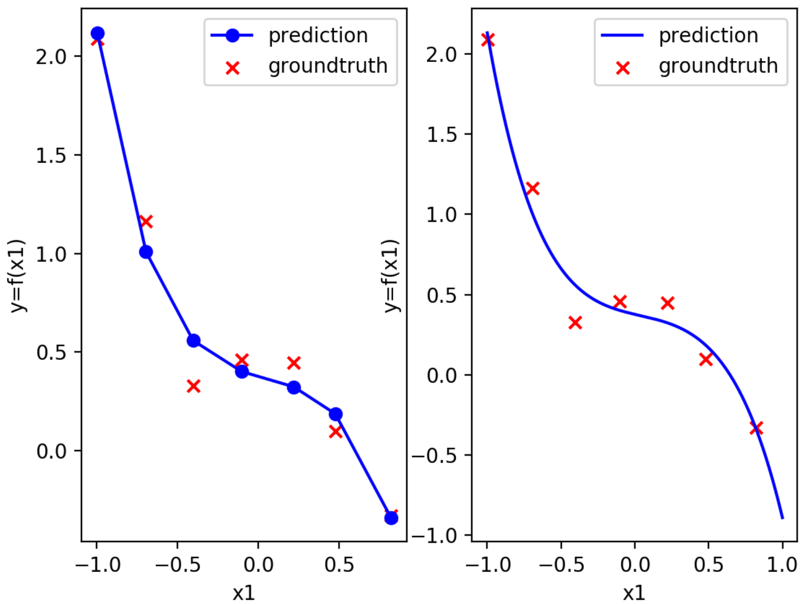

Fig. 8: Hypothesis function, for $\alpha = 1.4$.

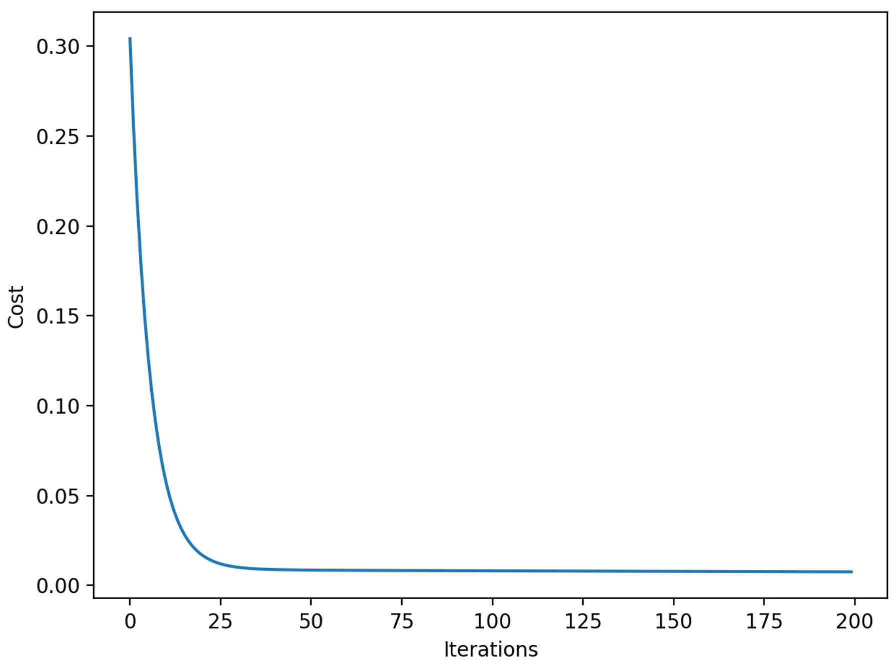

Fig. 9: Cost function by iteration, for $\alpha = 1.4$.

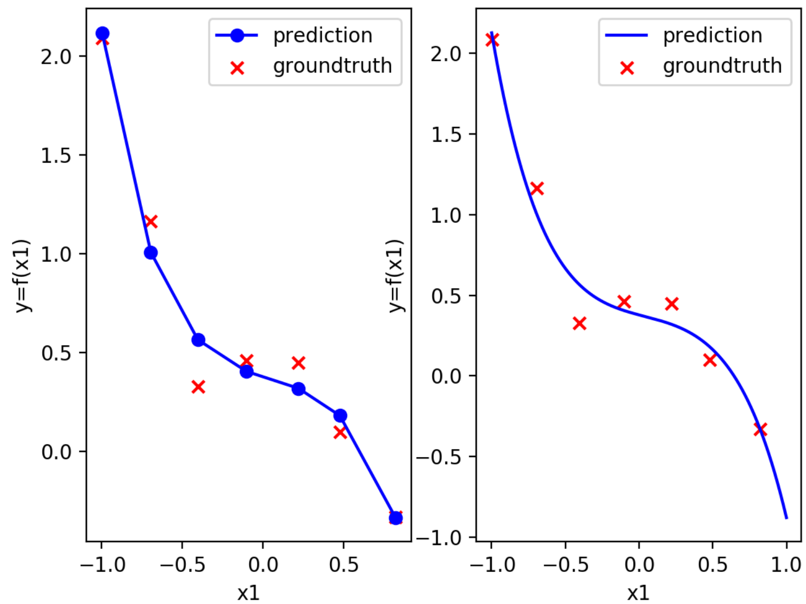

Fig. 10: Hypothesis function, for $\alpha = 1.4$ and $= 0.01$.

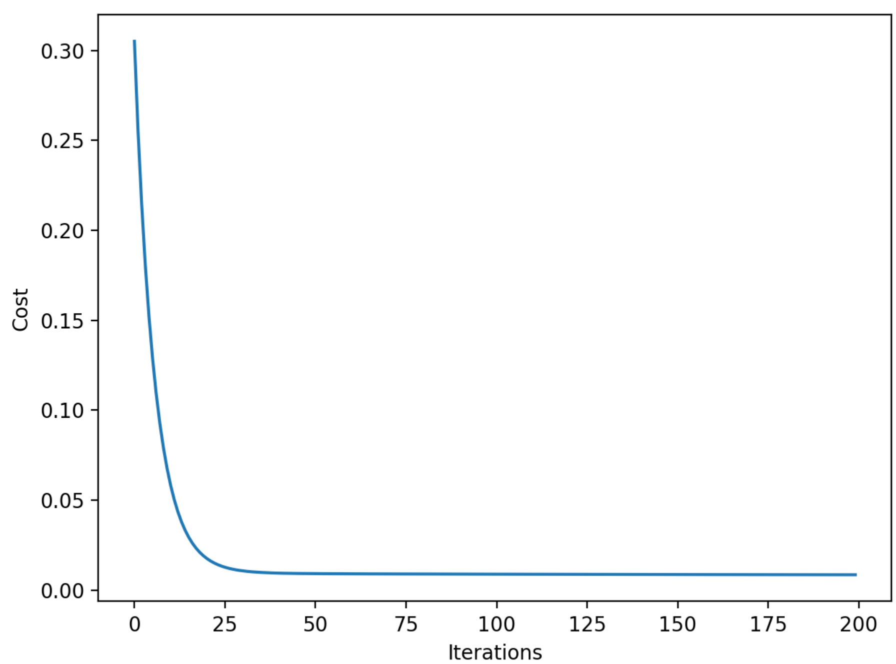

Fig. 11: Cost function by iteration, for $\alpha = 1.4$ and $= 0.01$.

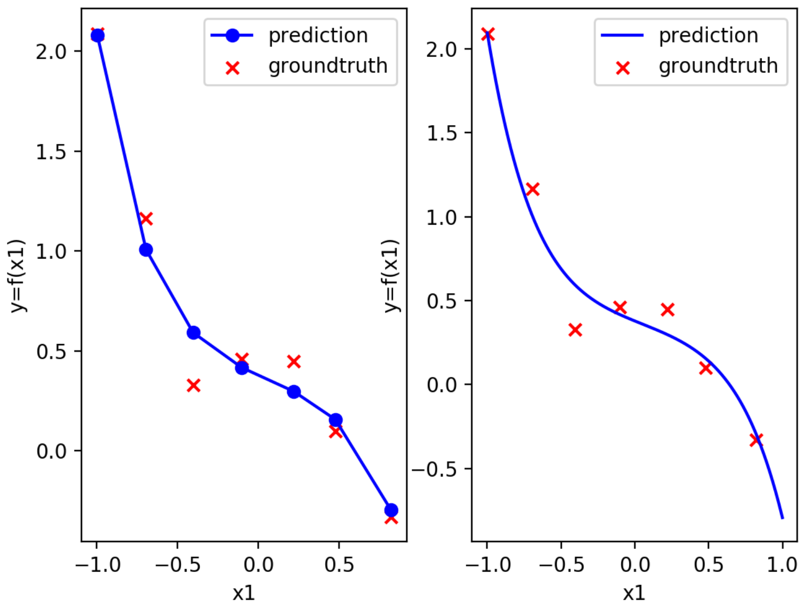

Fig. 12: Hypothesis function, for $\alpha = 1.4$ and $= 0.1$.

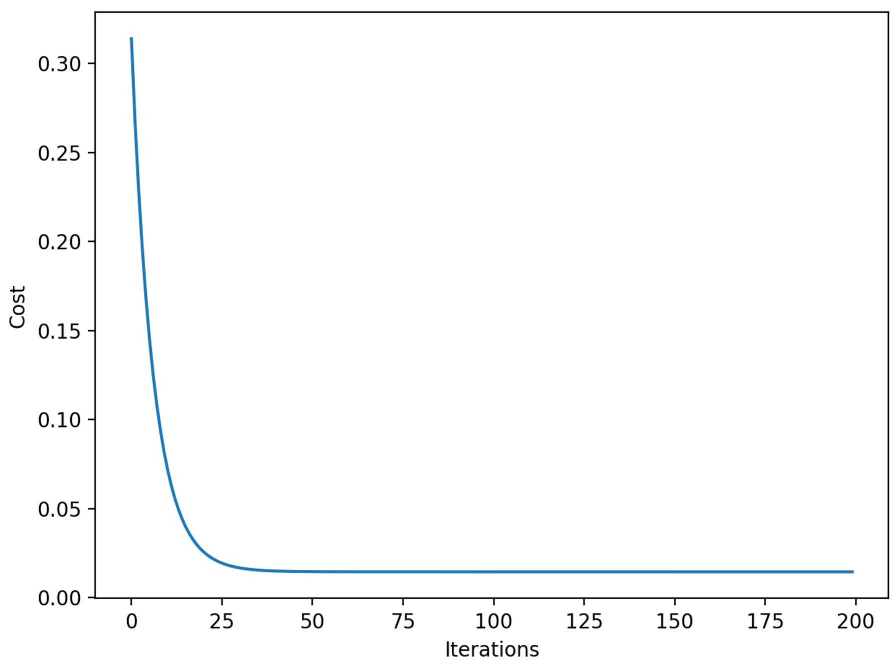

Fig. 13: Cost function by iteration, for $\alpha = 1.4$ and $= 0.1$.

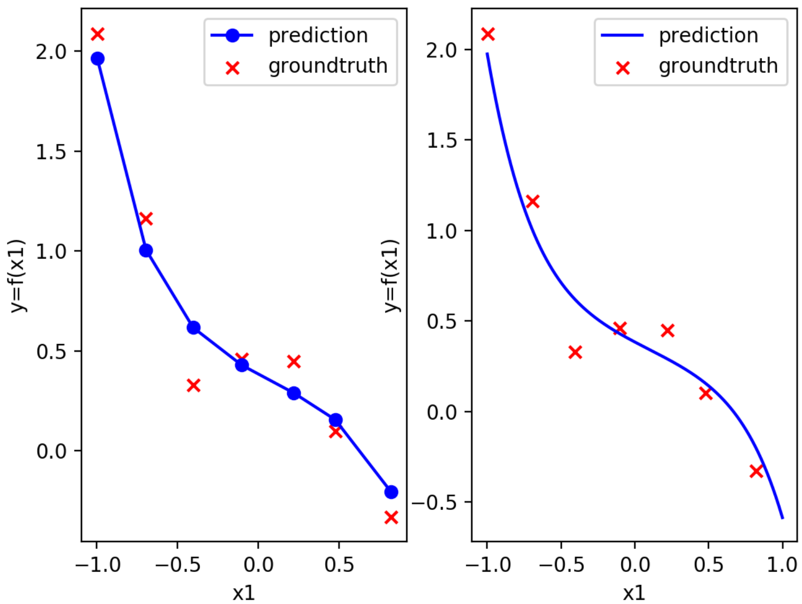

Fig. 14: Hypothesis function, for $\alpha = 1.4$ and $= 0.5$.

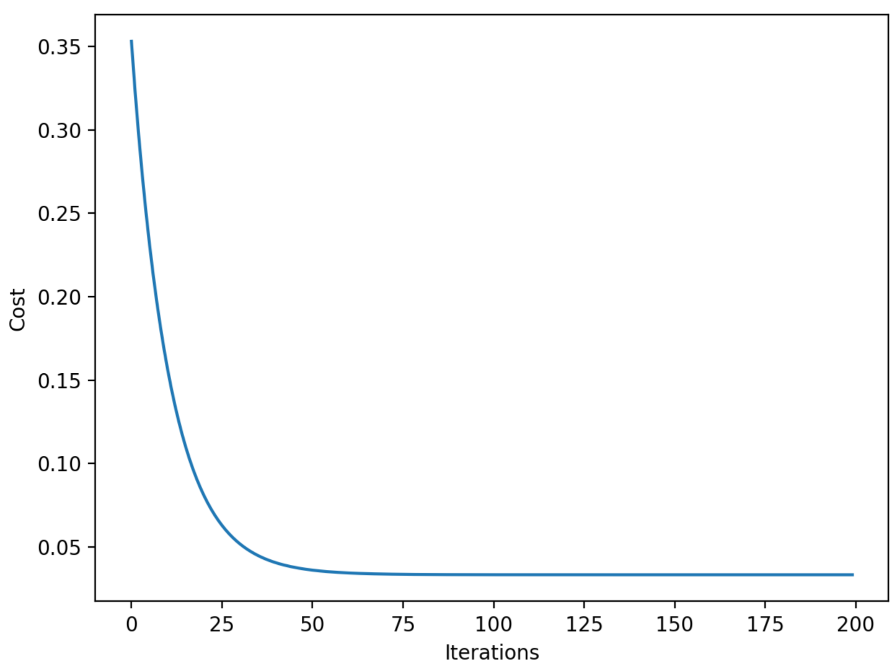

Fig. 15: Cost function by iteration, for $\alpha = 1.4$ and $= 0.5$.

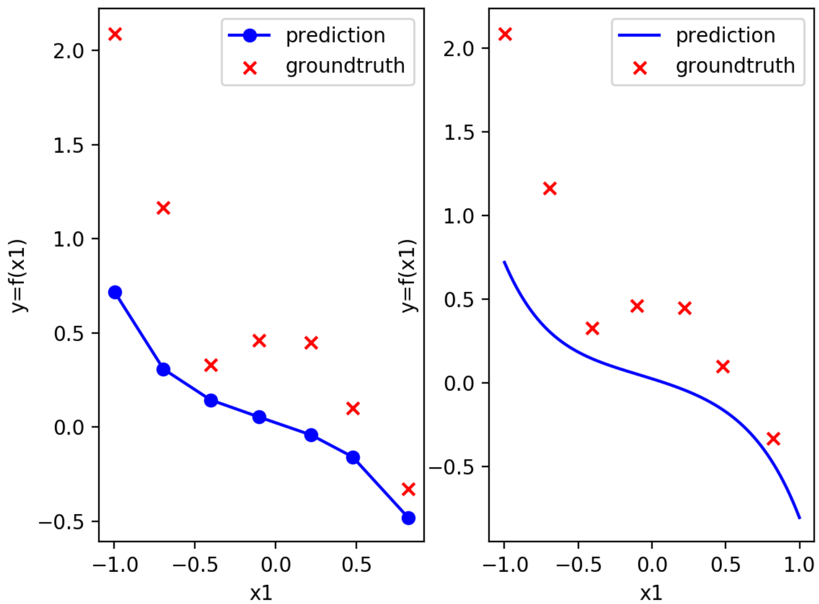

Fig. 16: Hypothesis function, for $\alpha = 1.4$ and $= 1$.

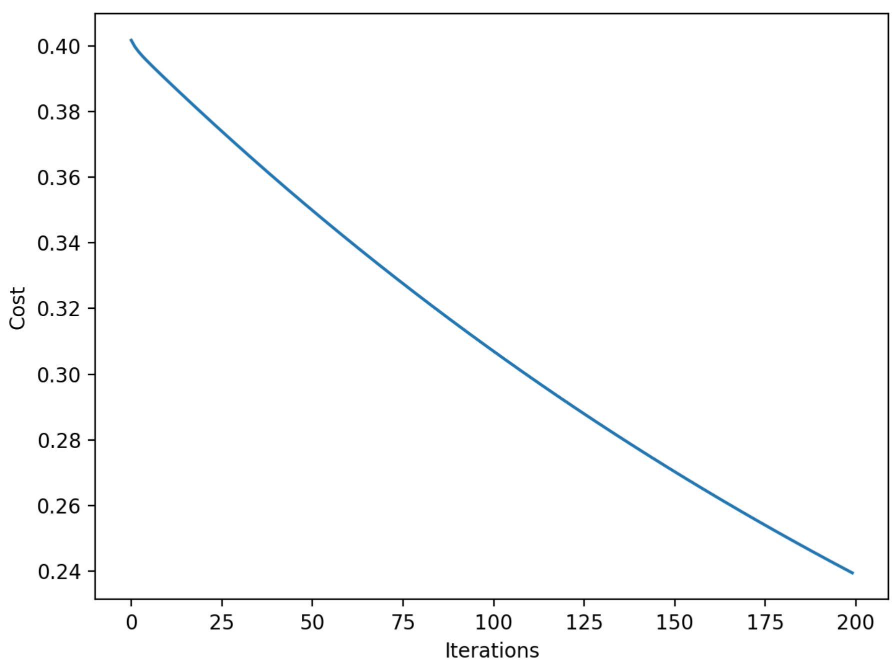

Fig. 17: Cost function by iteration, for $\alpha = 1.4$ and $= 1$.

As we can observe between Figure 10 and 18, increasing λ increases the cost function optimum, which was expected since we are just computing the cost against training data.