

---

## Homework #1

Due: 16th October 2023, Monday, before 11:59 pm

---

### Submission instructions

- Submit your solutions electronically on the course Gradescope site as PDF files.
- If you plan to typeset your solutions, please use the LaTeX solution template. If you must submit scanned handwritten solutions, please use a black pen on blank white paper and a high-quality scanner app.
- You are required to paste the snapshots of your codes with the title ‘Source Codes’ at the end of your solutions pdf.

## Problem 1 (SHORT ANSWER QUESTION)

[10 points total] For each of the statements below, state True (T) or False (F), or pick the correct option. Explain your answer in 1-2 sentences. For every part below, 1 point for correct assertion and 1 point for correct explanation.

- (a) T/F: Ridge regression does not have a closed-form solution.
  
- (b) T/F: If we have  $m$  values for a hyperparameter of a classifier and we use  $k$  fold cross-validation for hyperparameter tuning, then we train the classifier  $mk$  times from scratch.
  
- (c) T/F: Consider a linear function basis  $\phi(\mathbf{x}) = [1, \log x_1, x_2^3 x_3]$  where  $\mathbf{x} \in \mathbb{R}^3$ . There exists a closed form solution for minimizing the mean squared error for the hypothesis  $h_{\theta}(\mathbf{x}) = \theta^T \phi(\mathbf{x})$ . If True, state the solution. Otherwise, give a justification.
  
- (d) Which of the following is a major drawback of linear regression?
  - A. It assumes a linear relationship between the independent and dependent variables
  - B. It can handle only numerical data
  - C. It is not suitable for high-dimensional data
  - D. It is prone to overfitting
  - E. None of the above
  
- (e) Which of the following is a common method for preventing overfitting in machine learning?
  - A. Increasing the complexity of the model
  - B. Decreasing the size of the training data set
  - C. Regularization
  - D. Using a simpler evaluation metric
  - E. None of the above

## Problem 2 (MEAN ABSOLUTE ERROR)

[10 points total]

In class, we considered optimizing the Mean Square Error (MSE) loss. In practice, there are other choices of loss functions as well. For this problem, we will consider linear regression using Mean Absolute Error (MAE) as our loss function. Specifically, the MAE loss is given as:

$$J_{\text{MAE}}(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^n |\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)}|. \quad (1)$$

- (a) [4 points] Derive the partial derivative  $\frac{\partial J_{\text{MAE}}(\boldsymbol{\theta})}{\partial \theta_j}$ .

Hint: For this question, you don't need to worry about  $J = 0$ . You can use the vector-valued

$$\text{sign} \text{ function s.t. } \text{sign}(\hat{\mathbf{y}} - \mathbf{y}) = \begin{cases} +1 & \hat{y}^{(i)} \geq y^{(i)}, \\ -1 & \text{otherwise.} \end{cases}$$

- (b) [2 points] Write the vectorized solution for the gradient of the loss function, i.e.,  $\nabla_{\boldsymbol{\theta}} J_{\text{MAE}}(\boldsymbol{\theta})$ .

- (c) [4 points] Given the following dataset of 8 points (listed without bias feature  $x_0^{(i)}$ ), what is the value of  $J_{\text{MAE}}(\boldsymbol{\theta})$  and  $J_{\text{MSE}}(\boldsymbol{\theta})$  at  $\boldsymbol{\theta} = [1.0, 1.0, 1.0]^T$ ? How about their gradients  $\nabla_{\boldsymbol{\theta}} J_{\text{MAE}}(\boldsymbol{\theta})$  and  $\nabla_{\boldsymbol{\theta}} J_{\text{MSE}}(\boldsymbol{\theta})$ ?

$i$	1	2	3	4	5	6	7	8
$\mathbf{x}^{(i)}$	[4,0]	[1,1]	[0,1]	[-2,-2]	[-2,1]	[1,0]	[5,2]	[3,0]
$y^{(i)}$	12	3	1	6	3	6	8	7

### Problem 3 (PROGRAMMING EXERCISE: POLYNOMIAL REGRESSION)

[34 points total] In this exercise, you will work through linear and polynomial regression. Our data consists of (scalar) inputs  $x^{(i)} \in \mathbb{R}$  and outputs  $y^{(i)} \in \mathbb{R}, i \in \{1, \dots, n\}$ , which are related through a target function  $y^{(i)} = f(x^{(i)})$ . Your goal is to learn a linear predictor  $h_{\theta}(x)$  that best approximates  $f(x)$ .

---

code and data

- code : `regression.py`, `Notebook.ipynb`
  - data : `regression_train.csv`, `regression_valid.csv`, `regression_test.csv`
- 

### Visualization

As we learned last week, it is often useful to understand the data through visualizations. For this data set, you can use a scatter plot to visualize the data since it has only two properties to plot ( $x$  and  $y$ ).

- (a) [2 points] Visualize the training and test data using the `plot_data(...)` function. Simply by looking at the dataset, can you make an educated guess on the effectiveness of linear regression in predicting the label  $y$  using the default feature  $x$ ?

### Linear Regression

Recall that linear regression attempts to minimize the objective function

$$J(\theta) = \frac{1}{2n} \sum_{i=1}^n \left( h_{\theta}(\mathbf{x}^{(i)}) - y^{(i)} \right)^2.$$

In this problem, we will use the matrix-vector form where

$$\mathbf{y} = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{pmatrix}, \quad \mathbf{X} = \begin{pmatrix} \mathbf{x}^{(1)T} \\ \mathbf{x}^{(2)T} \\ \vdots \\ \mathbf{x}^{(n)T} \end{pmatrix}, \quad \boldsymbol{\theta} = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_d \end{pmatrix}$$

and each instance  $\mathbf{x}^{(i)} = (1, x_1^{(i)}, \dots, x_d^{(i)})^T$ .

Rather than working with this fully generalized, multivariate case, let us start by considering a simple linear regression model with  $d = 1$ . Similar to what we did in class, we first augment an extra dimension  $x_0 = 1$  to vectorize our hypothesis.

$$h_{\theta}(\mathbf{x}) = \boldsymbol{\theta}^T \mathbf{x} = \theta_0 + \theta_1 x_1$$

The file `regression.py` contains the skeleton code for the class `Regression`. Objects of this class can be instantiated as `model = Regression(m)` where  $m$  is the degree of the polynomial feature

vector we will touch upon soon. Polynomial feature vectors are a class of vectors having the form  $[1, x_1^{(i)}, x_1^{(i)^2}, \dots, x_1^{(i)^m}]^T$ . Setting  $m = 1$  instantiates a feature vector for instance  $i$  as  $[1, x_1^{(i)}]^T$ .

- (b) **[4 points]** Modify `get_poly_features()` in `Regression.py` for the case  $m = 1$  to create the matrix  $\mathbf{X}$  for a simple linear model. Include a screenshot of your code in the writeup.
- (c) **[4 points]** Before tackling the harder problem of training the regression model, complete `predict()` in `Regression.py` to predict  $\mathbf{y}$  from  $\mathbf{X}$  and  $\boldsymbol{\theta}$ . Include a screenshot of your code in the writeup.
- (d) In the lecture, we studied optimizing the loss for linear regression through gradient descent (GD) and its extension called mini-batch gradient descent (MBGD).

Recall that the parameters of our model are the  $\theta_j$  values. These are the values we will iteratively adjust to minimize  $J(\boldsymbol{\theta})$ . In MBGD, each iteration we sample a batch of  $B$  points  $D_B$  at random from the full dataset  $D$  without replacement and perform the update

$$\theta_j \leftarrow \theta_j - \alpha \frac{1}{B} \sum_{(\mathbf{x}^{(i)}, y^{(i)}) \in D_B} \left( h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)} \right) x_j^{(i)} \quad (\text{simultaneously update } \theta_j \text{ for all } j).$$

With each step of MBGD, we expect our updated parameters  $\theta_j$  to gradually come closer to the parameters that will achieve the lowest value of  $J(\boldsymbol{\theta})$ .

- **[4 points]** As we perform gradient descent, it is helpful to monitor the convergence by computing the loss, *i.e.*, the value of the objective function  $J$ . Complete `loss_and_grad()` to calculate  $J(\boldsymbol{\theta})$ , and the gradient. Complete `train_LR()` and plot the loss history.

We will use the following specifications for the gradient descent algorithm:

- We run the algorithm for 10,000 iterations.
- We will use a fixed step size.
- **[2 points]** So far, you have used a default learning rate (or step size) of  $\alpha = 0.01$ . Try different  $\alpha = 10^{-4}, 10^{-3}, 10^{-1}$ , and make a table of the learning rates and the final value of the objective function. Do all the learning rates lead to convergence?
- **[2 points]** Now let's fix  $\alpha = 0.01$ . Try different  $B = 1, 10, 20, n$ , where  $n$  is the size of the training dataset. Plot the loss history for different values of  $B$  on the same figure (*i.e.*, 4 curves in total for  $B = 1, 10, 20, n$ ).

- (e) **[4 points]** In class, we learned that the closed-form solution to linear regression is

$$\boldsymbol{\theta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}.$$

Using this formula, you will get an exact solution in one step. There is no iterative repetition of parameter updates like in gradient descent.

- Implement the closed-form solution `closed_form()`.
- Run your `closed_form()` to get solution. Report the result of parameters and the loss. How do they compare to those obtained by MBGD?

## Polynomial Regression

Now let us consider the more complicated case of polynomial regression. Here, we first define a polynomial feature map:  $\phi(x) = [1, x, x^2, \dots, x^m]^T$

Consequently, our hypothesis for linear regression is:

$$h_{\theta}(x) = \theta^T \phi(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \dots + \theta_m x^m.$$

Note, here we have NOT augmented our input  $x$  with an additional dimension, as done previously, as the vectorization can be induced directly after applying the feature map  $\phi$ .

- (f) **[6 points]** Recall that linear basis function regression (in this case, using a polynomial basis) can be considered as an extension of linear regression in which we replace our input matrix  $\mathbf{X}$  with

$$\Phi(\mathbf{X}) = \begin{pmatrix} \phi(x^{(1)})^T \\ \phi(x^{(2)})^T \\ \vdots \\ \phi(x^{(n)})^T \end{pmatrix},$$

where  $\phi(x)$  is a vector function such that  $\phi_j(x) = x^j$  for  $j = 0, \dots, m$ .

Update `get_poly_features()` for the case when  $m \geq 2$ . Submit a screenshot of your code for this question.

For  $m = \{0, \dots, 10\}$ , use the closed-form solver to determine the best-fit polynomial regression model on the training data, and with this model, calculate the loss on both the training data and the test data. Generate a plot depicting how loss varies with model complexity (polynomial degree) – you should generate a single plot with training, validation and test error, and include this plot in your writeup. Which degree polynomial would you say best fits the data? Was there evidence of under/overfitting the data? Use your plot to justify your answer.

## Regularization

Finally, we will explore the role of regularization. For this problem, we will use  $L_2$ -regularization so that our regularized objective function is

$$J(\theta) = \frac{1}{2n} \sum_{i=1}^n \left( h_{\theta}(\mathbf{x}^{(i)}) - y^{(i)} \right)^2 + \frac{\lambda}{2} \|\theta_{[1:m]}\|^2,$$

again optimizing for the parameters  $\theta$ .

- (g) **[6 points]** Modify `loss_and_grad()` to incorporate  $\ell_2$ -regularization using the closed-form solution. Submit a screenshot of your code for this question. Use your updated solver to find the coefficients that minimize the error for a tenth-degree polynomial ( $m = 10$ ) given regularization factor  $\lambda = 0, 10^{-8}, 10^{-7}, \dots, 10^{-1}, 10^0$ . Now we want to check how  $\lambda$  affects the loss (unregularized) on the training data, validation data, and test data. Generate a plot depicting how the loss error varies with  $\lambda$  (for your x-axis, let  $x = [1, 2, \dots, 10]$  correspond to  $\lambda = [0, 10^{-8}, 10^{-7}, \dots, 10^0]$  so that  $\lambda$  is on a logarithmic scale, with regularization increasing as  $x$  increases), and include this plot in your writeup. Which  $\lambda$  value appears to work best?