CS 446 / ECE 449 — Homework 2

your NetID here

Version 2.0

Instructions.

- Homework is due **Wednesday, September 28**, at 11:59 **AM** CST; you have **3** late days in total for **all Homeworks**.
- The template for coding problems are available at this link.
- Everyone must submit individually at gradescope under `Homework 2` and `Homework 2 Code`.
- The "written" submission at `Homework 2` **must be typed**, and submitted in any format gradescope accepts (to be safe, submit a PDF). You may use LaTeX, markdown, google docs, MS word, whatever you like; but it must be typed!
- When submitting at `Homework 2`, gradescope will ask you to m**ark out boxes around each of your answers**; please do this precisely!
- Please make sure your NetID is clear and large on the first page of the homework.
- Your solution **must** be written in your own words. Please see the course webpage for full **academic integrity** information. You should cite any external reference you use.
- We reserve the right to reduce the auto-graded score for `Homework 2 Code` if we detect funny business (e.g., your solution lacks any algorithm and hard-codes answers you obtained from someone else, or simply via trial-and-error with the autograder).
- When submitting to `Homework 2 Code`, only upload `hw2.py`. Additional files will be ignored.

Version History.

1. Initial version.
2. Updated for Fall 2022.

1. Miscellaneous.

(a) **Robustness of Linear Regression.** Consider a linear regression problem with a dataset containing N data points $\{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$, where $\boldsymbol{x}^{(i)} \in \mathbb{R}^d$. The loss function is given by:

$$\ell(\boldsymbol{w}) = \sum_{i=1}^{N} (y^{(i)} - \boldsymbol{w}_a^\intercal \boldsymbol{x}^{(i)} - w_b)^2$$

where $\boldsymbol{w} =< \boldsymbol{w}_a, w_b >$. Let's consider a particular 1-dimensional linear regression problem, where $\boldsymbol{x}^{(i)} \in \mathbb{R}^1$ and $\boldsymbol{w}_a \in \mathbb{R}^1$ are real numbers. Let's also fix $w_b = 1$.

i. Given a dataset $\{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{5} = \{(0,1),(1,2),(2,3),(3,4),(4,5)\}$, solve for $\boldsymbol{w}_a$.

ii. Give this dataset an unreasonable outlier $\{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{5} = \{(0,1),(1,2),(2,3),(3,4),(4,5),(5,100)\}$, solve for $\boldsymbol{w}_a$.

iii. Let's use L_1 norm for the loss function $\ell(\boldsymbol{w}) = \sum_{i=1}^{N} \|y^{(i)} - \boldsymbol{w}_a^\intercal \boldsymbol{x}^{(i)} - w_b\|_1$. Given the dataset with outlier $\{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{5} = \{(0,1),(1,2),(2,3),(3,4),(4,5),(5,100)\}$, solve for $\boldsymbol{w}_a$.

(b) **Logistic Regression.** The multinomial logistic classifier can use the softmax function to compute $p(Y = k|X)$ for $k = 1, \ldots, C$. The softmax function takes C arbitrary values and maps them to a probability distribution, with each value in the range $(0, 1)$, and all the values summing to 1. Assume we have C different classes, and posterior probability for class k is given by:

$$P(Y = k|X = \boldsymbol{x}) = \frac{\exp(\boldsymbol{w}_k^\intercal \boldsymbol{x})}{\sum_{c=1}^{C} \exp(\boldsymbol{w}_c^\intercal \boldsymbol{x})},$$

where $\boldsymbol{w}_i$ is the parameter. Assume that you are given N training examples $\{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$. Our goal is to estimate the weights using gradient ascent.

i. What is the data conditional log likelihood $L(\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_C)$.
$L(\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_C) = \log \prod_{i=1}^{N} P(y^{(i)}|\boldsymbol{x}^{(i)}, \boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_C)$

ii. Derive the partial derivative of $L(\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_C)$ with respect to $\boldsymbol{w}_k$.

(c) **Kernel.** We have two different definitions of kernels: Given any two data points $\boldsymbol{x}^{(i)}$ and $\boldsymbol{x}^{(j)}$,
Definition 1: $\kappa(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})$ is a kernel if it can be written as an inner product $\phi(\boldsymbol{x}^{(i)})^\intercal \phi(\boldsymbol{x}^{(j)})$ for some feature mapping $\boldsymbol{x} \to \phi(\boldsymbol{x})$.
Definition 2: $\kappa(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})$ is a kernel if for any finite set of training examples, $\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(N)}$, the $N \times N$ matrix $\boldsymbol{K}$ such that $\boldsymbol{K}_{ij} = \kappa(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})$ is positive semi-definite.

i. Show that Definition 1 implies Definition 2.

ii. Show that Definition 2 implies Definition 1.

Solution.

Your solution here.

2. Programming - Linear Regression.

The empirical risk in the linear regression method is defined as

$$\widehat{\mathcal{R}}(w_0, \boldsymbol{w}) := \frac{1}{N} \sum_{i=1}^{N} \left(\boldsymbol{w}^\top \boldsymbol{x}^{(i)} + w_0 - y^{(i)} \right)^2,$$

where $\boldsymbol{x}^{(i)} \in \mathbb{R}^d$ is a data point, $\boldsymbol{w}$ is a length-d vector with elements $(w_1, w_2, \cdots, w_d)^\top$ (in lecture we are using $\boldsymbol{w}_a$ for this term), w_0 is the bias term (in lecture we are using w_b for this term), and $y^{(i)}$ is an associated label.

For simplicity, we define $\boldsymbol{w}' = (w_0, \boldsymbol{w}^\top)^\top = (w_0, w_1, w_2, \cdots, w_d)^\top$, a length-$d+1$ vector that **prepends** w_0 to $\boldsymbol{w}$. $\boldsymbol{w}'$ stands for the **parameters** for this problem. We also define $\widehat{\mathcal{R}}(\boldsymbol{w}') = \widehat{\mathcal{R}}(w_0, \boldsymbol{w})$.

(a) **Implement linear regression using gradient descent in the `linear_gd(X, Y, lrate, num_iter)`.**

The objective of this function is to find parameters $\boldsymbol{w}'$ that minimize the empirical risk $\widehat{\mathcal{R}}(\boldsymbol{w}')$ using gradient descent (only gradient descent).

The arguments for this function are:

- `X` as the training features, a tensor with shape $N \times d$;
- `Y` as the training labels, an $N \times 1$ tensor;
- `lrate` as learning rate, a float number; and
- `num_iter` as the number of iterations for gradient descent to run, an integer.
- The **return value** is $\boldsymbol{w}'$, an $(d+1) \times 1$ tensor, the resulting parameter after gradient descending.

Implementation Instructions:

- Please use $\boldsymbol{w}' = 0$ as the initial parameters.

Note. The autograder evaluates your code using `FloatTensor`s for all computations. If you use `DoubleTensor`s, your results will not match those of the autograder due to the higher precision, and you may fail the tests in this case. PyTorch constructs `FloatTensor`s by default, so simply don't explicitly convert your tensors to `DoubleTensor`s or change the default tensor.

Hint. Suggestions about PyTorch:

- Try using the batched vector/matrix operations provided in PyTorch (like `torch.sum`) and avoid using for-loops.
  This will improve both the efficiency (by surprising 100 times!) and style of your program.
- Create your own test cases for debugging before submission. With very few samples in your own test case, it is convenient to compare the program output with your manual calculation.
- To avoid matrix computation and other tensor errors, remember to check the shapes of tensors regularly.

Hint. If you are not familiar with PyTorch but are familiar with NumPy, you can use `x.numpy()` to convert the inputs to NumPy, work with NumPy arrays, and at last use `torch.from_numpy(x)` to convert it back to PyTorch for returning. Still, you should try to use batched operations in NumPy (like `np.sum`) for higher efficiency. **However, you are highly recommended to get familiar with PyTorch, since there will be PyTorch-only homework in the future.**

Library routines: `torch.matmul (@), torch.tensor.shape, torch.tensor.t, torch.cat, torch.ones, torch.zeros, torch.reshape`.

(b) **Implement linear regression by using the pseudo inverse to solve for $\boldsymbol{w}'$ in the `linear_normal(X,Y)` function.**

In the lecture, we are told that we can solve linear regression not only by gradient descending like subproblem (a), but we can also use pseudo inverse to directly derive a closed-form optimal solution.

In this problem, similar to w', if you define $X' = (\mathbf{1}^{n \times 1}, X)$ (prepending a column of ones to X), then $\widehat{\mathcal{R}}(\boldsymbol{w}')$ can be simply written as

$$\widehat{\mathcal{R}}(\boldsymbol{w}') = \frac{1}{N} \sum_{i=1}^{N} \left(\boldsymbol{w}'^\top \boldsymbol{x}'^{(i)} - y^{(i)} \right)^2,$$

which can be solved with closed form

$$\arg\max_{\boldsymbol{w}'} \widehat{\mathcal{R}}(\boldsymbol{w}') = \textbf{pseudo-inverse}(X')\boldsymbol{y}.$$

In this function, you should implement linear regression solver by using pseudo inverse.

The arguments for this function are:

- `X` as the training features, a tensor with shape $N \times d$ tensor;
- `Y` as the training labels, an $N \times 1$ tensor.
- The **return value** is $\boldsymbol{w}'$, an $(d+1) \times 1$ tensor, the resulting parameter solved with pseudo inverse, i.e. $\arg\max_{\boldsymbol{w}'} \widehat{\mathcal{R}}(\boldsymbol{w}')$.

Library routines: `torch.matmul (@), torch.cat, torch.ones, torch.pinverse`.

(c) **Implement the `plot_linear()` function.** Follow the steps below.

- Use the provided function `hw2_utils.load_reg_data()` to generate a training set `X` and training labels `Y`.
- Then use `linear_normal()` to calculate the regression results $\boldsymbol{w}'$.
- Plot the points of dataset and regressed curve.
- Return the plot (figure object, `plt.gcf()`) as the output.

You should include the visualization **in your written submission**.

Hint. If you are new to plotting machine learning visualizations, we offer some kind suggestions. `matplotlib.pyplot` is an "extremely" useful tool in machine learning, and we commonly refer to it as `plt`. Please first get to know the most basic usages by examples from its official website (such as scatter plots, line plots, etc.). As for our programming question specifically, you may divide and conquer it by first plotting the points in the dataset, then plotting the linear regression curve.

Library routines: `torch.matmul (@), torch.cat, torch.ones, torch.flatten` (might be useful in `plt.scatter`), `plt.plot, plt.scatter, plt.show, plt.gcf`.
Where `plt` refers to the `matplotlib.pyplot` library.

Solution.

Your solution here.

3. Programming - Logistic Regression.

The empirical risk $\widehat{\mathcal{R}}$ for logistic regression:

$$\widehat{\mathcal{R}}_{\log}(\boldsymbol{w}') = \frac{1}{N}\sum_{i=1}^{N} \log\left(1 + \exp\left(-y^{(i)}\left(\boldsymbol{w}^\top \boldsymbol{x}^{(i)} + w_0\right)\right)\right).$$

Here you will minimize this risk using gradient descent. We have similar definitions for $\boldsymbol{w}', \boldsymbol{w}, w_0$ as Problem 2.

(a) In your **written submission**, derive the gradient descent update rule for this empirical risk by taking the gradient. Write your answer in terms of the learning rate η, previous parameters $\boldsymbol{w}'$, new parameters $\boldsymbol{w}'_{\text{new}}$, number of examples N, and training examples $\boldsymbol{x}^{(i)}$. Show all of your steps.

(b) **Implement the `logistic(X, Y, lrate, num_iter)` function.**

You are given as input a training set `X`, training labels `Y`, a learning rate `lrate`, and number of gradient updates `num_iter`. The format is the same as Problem 2 (a).

Implement gradient descent to find parameters $\boldsymbol{w}'$ that minimize the empirical risk $\widehat{\mathcal{R}}_{\log}(\boldsymbol{w}')$. Perform gradient descent for `num_iter` updates with a learning rate of `lrate`.

Same as Problem 2 (a), initialize $\boldsymbol{w}' = 0$, return $\boldsymbol{w}'$ as output.

Library routines: `torch.matmul (@), torch.tensor.t, torch.exp.`

(c) **Implement the `logistic_vs_ols()` function.**

In this function, you should:

- Use `hw2_utils.load_logistic_data()` to generate a training set `X` and training labels `Y`.
- Run `logistic(X,Y)` from subproblem (b) taking `X` and `Y` as input to obtain parameters $\boldsymbol{w}'$ (use the defaults for `num_iter` and `lrate`).
- Also run `linear_gd(X,Y)` from Problem 2 to obtain parameters $\boldsymbol{w}'$.
- Plot the decision boundaries for your logistic regression and least squares models along with the data `X`.
- Return the plot (figure object, `plt.gcf()`) as the output.

Note. As we learned in the class that the decision boundary of Logistic Regression can be obtained from $\widehat{\boldsymbol{w}}^\top \boldsymbol{x} + \widehat{w}_0 = 0$, i.e., for $d = 2$, we have $x_2 = -(\widehat{w}_0 + \widehat{w}_1 x_1)/\widehat{w}_2$.

Include the visualizations in your **written submission**.

Which model appears to classify the data better? Explain in the **written submission** that why you believe your choice is the better classifier for this problem.

Library routines: `torch.linspace, plt.scatter, plt.plot, plt.show, plt.gcf.`

Solution.

Your solution here.

4. Learning Theory.

(a) **VC Dimensions.** In this problem, we'll explore VC dimensions! First, a few definitions that we will use in this problem. For a feature space $\mathcal{X}$, let $\mathcal{F}$ be a set of binary classifier of the form $f : \mathcal{X} \to \{0, 1\}$. $\mathcal{F}$ is said to **shatter** a set of r distinct points $\{\boldsymbol{x}^{(i)}\}_{i=1}^{r} \subset \mathcal{X}$ if for each set of label assignments $(y^{(i)})_{i=1}^{r} \in \{0,\, 1\}^r$ to these points, there is an $f \in \mathcal{F}$ which makes no mistakes when classifying D.

The VC Dimension of $\mathcal{F}$ is the largest non-negative integer $r \in$ such that there is a set of r points that $\mathcal{F}$ can shatter. Even more formally, let $VC(\mathcal{F})$ denote the VC Dimension of $\mathcal{F}$. It can be defined as:

$$VC(\mathcal{F}) = \max_{r} \; r \qquad \text{s.t. } \exists \{\boldsymbol{x}^{(i)}\}_{i=1}^{r} \subset \mathcal{X}, \forall (y^{(i)})_{i=1}^{r} \in \{0,\, 1\}^r \;, \exists f \in \mathcal{F}, \forall i : f(\boldsymbol{x}^{(i)}) = y^{(i)}$$

The intuition here is that VC dimension captures some kind of complexity or capacity of a set of functions $\mathcal{F}$.

Note: The straightforward proof strategy to show that the VC dimension of a set of classifiers is r is to first show that for a set of r points, the set is shattered by the set of classifiers. Then, show that any set of $r + 1$ points cannot be shattered. You can do that by finding an assignment of labels which cannot be correctly classified using $\mathcal{F}$.

Notation: We denote $\mathbb{I}_{\text{condition}}(\cdot) : \mathcal{X} \to \{0, 1\}$ to be the indicator function, i.e., $\mathbb{I}_{\text{condition}}(x) = 1$ if the condition is true for x and $\mathbb{I}_{\text{condition}}(x) = 0$ otherwise.

We will now find the VC dimension of some basic classifiers.

i. **Upper Bound**
For any finite set of binary classifiers $\mathcal{F}$, prove that $VC(\mathcal{F}) \leq \log_2 |\mathcal{F}|$

ii. **1D Affine Classifier**
Let's start with a fairly simple problem. Consider $\mathcal{X} = \mathbb{R}$ and $\mathcal{Y} = \{0, 1\}$. Affine classifiers are of the form:

$$\mathcal{F}_{\text{affine}} = \{\mathbb{I}_{wx+b\geq 0}(\cdot) : \mathcal{X} \to \mathbb{R} \mid w, b \in \mathbb{R}\},$$

Show what is $VC(\mathcal{F}_{\text{affine}})$ and prove your result.
Hint: Try less than a handful of points.

iii. **General Affine Classifier**
We will now go one step further. Consider $\mathcal{X} = \mathbb{R}^d$ for some dimensionality $d \geq 1$, and $\mathcal{Y} = \{0, 1\}$. Affine classifiers in d dimensions are of the form

$$\mathcal{F}^{k}_{\text{affine}} = \{\mathbb{I}_{\boldsymbol{w}^\top \boldsymbol{x}+b\geq 0}(\cdot) : \mathcal{X} \to \mathbb{R} \mid \boldsymbol{w} \in \mathbb{R}^d, b \in \mathbb{R}\}$$

Show what is $VC(\mathcal{F}^{d}_{\text{affine}})$ and prove your result.
Hint: Note that $\boldsymbol{w}^\top \boldsymbol{x} + b$ can be written as $[\boldsymbol{x}^\top \;\; 1] \begin{bmatrix} \boldsymbol{w} \\ b \end{bmatrix}$. Moreover, consider to put all data points into a matrix, e.g.,

$$\boldsymbol{X} = \begin{bmatrix} (\boldsymbol{x}^{(1)})^\top & 1 \\ (\boldsymbol{x}^{(2)})^\top & 1 \\ \vdots & \vdots \end{bmatrix}.$$

(b) **Rademacher Complexity.** Recall from class that the generalization error bound scales with the complexity of the function class $\mathcal{F}$, which, in turn, can be measured via Rademacher complexity. In this question we will compute the Rademacher complexity of linear functions step by step. Let's consider a dataset $\{\boldsymbol{x}^{(i)}\}_{i=1}^{N} \subset \mathbb{R}^d$ with the norm bounded by $\|\boldsymbol{x}^{(i)}\|_2 \leq R$ and the set of linear classifiers $\mathcal{F} = \{\boldsymbol{x} \mapsto \boldsymbol{w}^\top \boldsymbol{x} \;\mid \boldsymbol{w} \in \mathbb{R}^d, \|\boldsymbol{w}\|_2 \leq W\}$.

i. For a fixed sign vector $\boldsymbol{\epsilon} = (\epsilon_1, ..., \epsilon_N) \in \{\pm 1\}^N$ show that:

$$\max_{f \in \mathcal{F}} \frac{1}{N} \sum_{i=1}^{N} \epsilon_i f(\boldsymbol{x}^{(i)}) \leq W \|\boldsymbol{x}_{\boldsymbol{\epsilon}}\|_2$$

where $\boldsymbol{x}_{\boldsymbol{\epsilon}}$ is defined as $\boldsymbol{x}_{\boldsymbol{\epsilon}} = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{x}^{(i)}\epsilon_i$.
Hint: Cauchy-Schwarz inequality.

ii. Assume ϵ_i is distributed i.i.d. according to $\Pr[\epsilon_i = +1] = \Pr[\epsilon_i = -1] = 1/2$. Show that

$$\mathbb{E}_{\boldsymbol{\epsilon}}\left[\|\boldsymbol{x}_{\epsilon}\|^2\right] \leq \frac{R^2}{N}$$

iii. Assume ϵ_i follows the same distribution as previous problem. Recall the definition of Rademacher complexity:

$$\mathrm{Rad}(\mathcal{F}) = \mathbb{E}_{\boldsymbol{\epsilon}}\left[\max_{f\in\mathcal{F}} \frac{1}{N}\sum_{i=1}^{N} \epsilon_i f(\boldsymbol{x}^{(i)})\right]$$

Show that the Rademacher complexity of the set of linear classifiers is:

$$\mathrm{Rad}(\mathcal{F}) \leq \frac{RW}{\sqrt{N}}$$

Hint: Jensen's inequality.

Solution.

Your solution here.