CS 446 / ECE 449 — Homework 4

your NetID here

Version 1.0

Instructions.

- Homework is due **Tuesday, April 4, at noon CST**; no late homework accepted.
- Everyone must submit individually at gradescope under `hw4` and `hw4code`.
- The "written" submission at `hw4` **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 `hw4`, gradescope will ask you to mark out boxes/pages 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. Briefly, you may have high-level discussions with at most 3 classmates, whose NetIDs you should place on the first page of your solutions, and you should cite any external reference you use; despite all this, your solution must be written in your own words.
- We reserve the right to reduce the auto-graded score for `hw4code` 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).
- Coding problems come with suggested "library routines"; we include these to reduce your time fishing around APIs, but you are free to use other APIs.
- When submitting to `hw4code`, only upload the two python files `hw4.py` and `hw4_utils.py`. Don't upload a zip file or additional files.

Version history.

1. Initial version.

1. Linear Regression/SVD.

Throughout this problem let $\boldsymbol{X}$ be the $n \times d$ matrix with the feature vectors $(\boldsymbol{x}_i)_{i=1}^n$ as its rows. Suppose we have the singular value decomposition $\boldsymbol{X} = \sum_{i=1}^r s_i \boldsymbol{u}_i \boldsymbol{v}_i^\top$.

(a) Let the training examples $(\boldsymbol{x}_i)_{i=1}^n$ be the standard basis vectors $\boldsymbol{e}_i$ of $\mathbb{R}^d$ with each $\boldsymbol{e}_i$ repeated $n_i > 0$ times having labels $\left(y_{i_j}\right)_{j=1}^{n_i}$. That is, our training set is:

$$\bigcup_{i=1}^{d} \left\{ \left(\boldsymbol{e}_i, y_{i_j}\right) \right\}_{j=1}^{n_i},$$

where $\sum_{i=1}^d n_i = n$. Show that for a vector $\boldsymbol{w}$ that minimizes the empirical risk, the components w_i of $\boldsymbol{w}$ are the averages of the labels $\left(y_{i_j}\right)_{j=1}^{n_i}$: $w_i = \frac{1}{n_i} \sum_{j=1}^{n_i} y_{i_j}$.

Hint: Write out the expression for the empirical risk with the squared loss and set the gradient equal to zero.

Remark: This gives some intuition as to why "regression" originally meant "regression towards the mean."

(b) Returning to a general matrix $\boldsymbol{X}$, show that if the label vector $\boldsymbol{y}$ is a linear combination of the $\{\boldsymbol{u}_i\}_{i=1}^r$ then there exists a $\boldsymbol{w}$ for which the empirical risk is zero (meaning $\boldsymbol{X}\boldsymbol{w} = \boldsymbol{y}$).

Hint: Either consider the range of $\boldsymbol{X}$ and use the SVD, or compute the empirical risk explicitly with $\boldsymbol{y} = \sum_{i=1}^r a_i \boldsymbol{u}_i$ for some constants a_i and $\hat{\boldsymbol{w}}_{\text{ols}} = \boldsymbol{X}^+ \boldsymbol{y}$.

Remark: It's also not hard to show that if $\boldsymbol{y}$ is not a linear combination of the $\{\boldsymbol{u}_i\}_{i=1}^r$, then the empirical risk must be nonzero.

(c) Show that $\boldsymbol{X}^\top \boldsymbol{X}$ is invertible if and only if $(\boldsymbol{x}_i)_{i=1}^n$ spans $\mathbb{R}^d$.

Hint: Recall that the squares of the singular values of $\boldsymbol{X}$ are eigenvalues of $\boldsymbol{X}^\top \boldsymbol{X}$.

Remark: This characterizes when linear regression has a unique solution due to the normal equation (note that we always have at least one solution obtained by the pseudoinverse). We would not have had a unique solution for part (a) if we had an $n_i = 0$.

(d) Provide a matrix $\boldsymbol{X}$ such that $\boldsymbol{X}^\top \boldsymbol{X}$ is invertible and $\boldsymbol{X}\boldsymbol{X}^\top$ is not. Include a formal verification of this for full points.

Hint: Use part (c). It may be helpful to think about conditions under which a matrix is not invertible.

Hint: The "formal verification" doesn't need to be a lengthy proof: in particular, there exist choices of $\boldsymbol{X}$ which are easy to write down, for which a verification of this claim is immediate. It can also be beneficial to *write down* the matrix directly via its SVD.

Solution.

(a)

(b)

(c)

(d)

2. Linear Regression.

Recall that the empirical risk in the linear regression method is defined as $\widehat{\mathcal{R}}(\boldsymbol{w}) := \frac{1}{2n}\sum_{i=1}^{n}(\boldsymbol{w}^\top \boldsymbol{x}_i - y_i)^2$, where $\boldsymbol{x}_i \in \mathbb{R}^d$ is a data point and y_i is an associated label.

(a) Implement linear regression using gradient descent in the `linear_gd(X, Y, lrate, num_iter)` function of `hw4.py`. You are given as input a training set `X` as an $n \times d$ tensor, training labels `Y` as an $n \times 1$ tensor, a learning rate `lrate`, and the number of iterations of gradient descent to run `num_iter`. Using gradient descent, find parameters $\boldsymbol{w}$ that minimize the empirical risk $\widehat{\mathcal{R}}(\boldsymbol{w})$. Use $\boldsymbol{w} = 0$ as your initial parameters, and return your final w as output. Prepend a column of ones to `X` in order to accommodate a bias term in $\boldsymbol{w}$.

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 pseudoinverse to solve for w in the `linear_normal(X,Y)` function of `hw4.py`. You are given a training set `X` as an $n \times d$ tensor and training labels `Y` as an $n \times 1$ tensor. Return your parameters w as output. As before, make sure to accommodate a bias term by prepending ones to the training examples `X`.

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

(c) Implement the `plot_linear()` function in `hw4.py`. Use the provided function `hw4_utils.load_reg_data()` to generate a training set `X` and training labels `Y`. Plot the curve generated by `linear_normal()` along with the points from the data set. Return the plot as output. Include the plot in your written submission.

Library routines: `torch.matmul (@), torch.cat, torch.ones, plt.plot, plt.scatter, plt.show, plt.gcf` where `plt` refers to the `matplotlib.pyplot` library.

Solution.

(c)

3. Polynomial Regression.

In Problem 3 you constructed a linear model $\boldsymbol{w}^\top \boldsymbol{x} = \sum_{i=1}^{d} x_i w_i$. In this problem you will use the same setup as in the previous problem, but enhance your linear model by doing a quadratic expansion of the features. Namely, you will construct a new linear model $f_{\boldsymbol{w}}$ with parameters

$$(w_0, w_{01}, \ldots, w_{0d}, w_{11}, w_{12}, \ldots, w_{1d}, w_{22}, w_{23}, \ldots, w_{2d}, \ldots, w_{dd})^\top,$$

defined by

$$f_{\boldsymbol{w}}(x) = \boldsymbol{w}^\top \phi(\boldsymbol{x}) = w_0 + \sum_{i=1}^{d} w_{0i} x_i + \sum_{i \le j}^{d} w_{ij} x_i x_j.$$

Warning: If the computational complexity of your implementation is high, it may crash the autograder (try to optimize your algorithm if it does)!

(a) Given a 3-dimensional feature vector $\boldsymbol{x} = (x_1, x_2, x_3)$, completely write out the quadratic expanded feature vector $\phi(\boldsymbol{x})$.

(b) Implement the `poly_gd()` function in `hw4.py`. The input is in the same format as it was in Problem 3. Implement gradient descent on this training set with $\boldsymbol{w}$ initialized to 0. Return $\boldsymbol{w}$ as the output with terms in this exact order: bias, linear, then quadratic. For example, if $d = 3$ then you would return $(w_0, w_{01}, w_{02}, w_{03}, w_{11}, w_{12}, w_{13}, w_{22}, w_{23}, w_{33})$.

Library routines: `torch.cat, torch.ones, torch.zeros, torch.stack.`

Hint: You will want to prepend a column of ones to `X`, and append to `X` the squared features in the specified order. You can generate the squared features in the correct order (This is important! The order of the polynomial features matters for your answer to match the correct answer on GradeScope. Check the order in the problem definition above.) using a nested for loop. We don't want duplicates (e.g., $x_0 x_1$ and $x_1 x_0$ should not both be included; we should only include $x_0 x_1$ in the quadratic case).

(c) Implement the `poly_normal` function in `hw4.py`. You are given the same data set as from part (b), but this time determine w by using the pseudoinverse. Return $\boldsymbol{w}$ in the same order as in part (b).

Library routines: `torch.pinverse.`

Hint: You will still need to transform the matrix `X` in the same way as in part (b).

(d) Implement the `plot_poly()` function in `hw4.py`. Use the provided function `hw4_utils.load_reg_data()` to generate a training set `X` and training labels `Y`. Plot the curve generated by `poly_normal()` along with the points from the data set. Return the plot as output and include it in your written submission. Compare and contrast this plot with the plot from Problem 3. Which model appears to approximate the data better? Justify your answer.

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

(e) The Minsky-Papert XOR problem is a classification problem with data set:

$$X = \{(-1, +1), (+1, -1), (-1, -1), (+1, +1)\}$$

where the label for a given point (x_1, x_2) is given by its product $x_1 x_2$. For example, the point $(-1, +1)$ would be given label $y = (-1)(1) = -1$. Implement the `poly_xor()` function in `hw4.py`. In this function you will load the XOR data set by calling the `hw4_utils.load_xor_data()` function, and then apply the `linear_normal()` and `poly_normal()` functions to generate predictions for the XOR points. Include a plot of contour lines that show how each model classifies points in your written submission. Return the predictions for both the linear model and the polynomial model and use `contour_plot()` in `hw4_utils.py` to help with the plot. Do both models correctly classify all points? (Note that red corresponds to larger values and blue to smaller values when using `contour_plot` with the "coolwarm" colormap).

Hint: A "Contour plot" is a way to represent a 3-dimensional surface in a 2-D figure. In this example, the data points are pined to the figure with their features (x_1, x_2) as the coordinates in 2-D space (e.g., x and y axis); the third dimension (e.g., the predictions of the data points) is labeled on the points in the figure. The lines or curves that link the grid points with the same predictions together are called the "contours". See `contour_plot()` in `hw4_utils.py` for details.

Solution.

(a)
(d)
(e)