CSCI-GA.2565 — Homework 1

your name and NetID here

Version 1.1

Instructions.

- **Due date.** Homework is due **Friday, February 14, at noon EST**.
- **Gradescope submission.** Everyone must submit individually at gradescope under `hw1` and `hw1code`: `hw1code` is just python code, whereas `hw1` contains everything else. For clarity, problem parts are annotated with where the corresponding submissions go.
  - **Submitting `hw1`.** `hw1` must be submitted as a single PDF file, and typeset in some way, for instance using LaTeX, Markdown, Google Docs, MS Word; you can even use an OCR package (or a modern multi-modal LLM) to convert handwriting to LaTeXand then clean it up for submission. Graders reserve the right to award zero points for solutions they consider illegible.
  - **Submitting `hw1code`.** Only upload the two python files `hw1.py` and `hw1_utils.py`; don't upload a zip file or additional files.
- **Consulting LLMs and friends.** You may discuss with your peers and you may use LLMs. *However,* you are strongly advised to make a serious attempt on all problems alone, and if you consult anyone, make a serious attempt to understand the solution alone afterwards. You must document credit assignment in a special final question in the homework.
- **Evaluation.** We reserve the right to give a 0 to a submission which violates the intent of the assignment and is morally equivalent to a blank response.
  - `hw1code:` your grade is what the autograder gives you; note that you may re-submit as many times as you like until the deadline. However, we may reduce your auto-graded score if your solution simply hard-codes answers.
  - `hw1:` you can receive 0 points for a blank solution, an illegible solution, or a solution which does not correctly mark problem parts with boxes in the gradescope interface (equivalent to illegibility). All other solutions receive full points, *however* the graders do leave feedback so please check afterwards even if you received a perfect score.
- **Regrades.** Use the grade scope interface.
- **Late days.** We track 3 late days across the semester per student.
- **Library routines.** 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.

Version history.

1.0. Initial version.

1.1. Fixed Q2 [`hw1`] and [`hw1code`] tags. Added refs.bib to the zip file.

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) [hw1] 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) [hw1] 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) [hw1] 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) [hw1] 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.

Solution.

2.

3. hello **Solution.**

4. 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) [`hw1code`] Implement linear regression using gradient descent in the `linear_gd(X, Y, lrate, num_iter)` function of `hw1.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) [`hw1code`] Implement linear regression by using the pseudoinverse to solve for w in the `linear_normal(X,Y)` function of `hw1.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) [`hw1`] Implement the `plot_linear()` function in `hw1.py`. Use the provided function `hw1_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.

5. Polynomial Regression.

In Problem 2 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) [`hw1`] 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) [`hw1code`] Implement the `poly_gd()` function in `hw1.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) [`hw1code`] Implement the `poly_normal` function in `hw1.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) [`hw1`] Implement the `plot_poly()` function in `hw1.py`. Use the provided function `hw1_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) [`hw1`] 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 `hw1.py`. In this function you will load the XOR data set by calling the `hw1_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 `hw1_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 `hw1_utils.py` for details.

Solution.

6. Logistic Regression.

Recall the empirical risk $\widehat{\mathcal{R}}$ for logistic regression (as presented in lecture 2):

$$\widehat{\mathcal{R}}_{\log}(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n} \ln(1 + \exp(-y_i \boldsymbol{w}^\top \boldsymbol{x}_i)).$$

Here you will minimize this risk using gradient descent.

(a) [`hw1`] 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 $\boldsymbol{w}'$, number of examples n, and training examples $(\boldsymbol{x}_i, y_i)$. Show all of your steps.

(b) [`hw1code`] Implement the `logistic()` function in `hw1.py`. You are given as input a training set `X`, training labels `Y`, a learning rate `lrate`, and number of gradient updates `num_iter`. 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`, initializing $\boldsymbol{w} = 0$ and returning $\boldsymbol{w}$ as output. Don't forget to prepend `X` with a column of ones.

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

(c) [`hw1`] Implement the `logistic_vs_ols()` function in `hw1.py`. Use `hw1_utils.load_logistic_data()` to generate a training set `X` and training labels `Y`. Run `logistic(X,Y)` from part (b) taking `X` and `Y` as input to obtain parameters $\boldsymbol{w}$. 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`. Which model appears to classify the data better? Explain why you believe your choice is the better classifier for this problem.

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

Hints:

- The positive and negative points are guaranteed to be linearly separable (though an algorithm may or may not find the optimal line to separate them).
- The "decision boundary" in the problem description refers to the set of points $\boldsymbol{x}$ such that $\boldsymbol{w}^\top \boldsymbol{x} = 0$ for the chosen predictor. In this case, it suffices to plot the corresponding line.
- In order to make the two models significantly different, we recommend that you train the logistic regression with a large `num_iter` (e.g., 1,000,000 or even larger).

Solution.

7. N-Gram Next Token Prediction.

Recall the empirical risk $\widehat{\mathcal{R}}$ for cross entropy (as presented in lecture 2):

$$\widehat{\mathcal{R}}_{\text{CE}}(\boldsymbol{W}) = -\frac{1}{n}\sum_{i=1}^{n} \ln\left(\frac{\exp(f_{y_i}(\boldsymbol{x}_i))}{\sum_{j=1}^{k}\exp(f_j(\boldsymbol{x}_i))}\right),$$

where for this problem we consider a linear predictor $f : \mathbb{R}^d \to \mathbb{R}^k$ given by $f(x) = \boldsymbol{W}^\top \boldsymbol{x}$. Here you will minimize this risk using gradient descent, and apply it to next token prediction.

(a) [hw1] 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 $\boldsymbol{W}'$, number of examples n, training examples $(\boldsymbol{x}_i, y_i)$, and probabilities $p(c|x_i) = \frac{\exp(f_c(\boldsymbol{x}_i))}{\sum_{j=1}^{k}\exp(f_j(\boldsymbol{x}_i))}$. Show all of your steps.

(b) [hw1code] Implement the `cross_entropy()` function in `hw1.py`. You are given as input a training set `X`, training labels `Y`, number of classes `k`, a learning rate `lrate`, and number of gradient updates `num_iter`. Implement gradient descent to find parameters $\boldsymbol{W}$ that minimize the empirical risk $\widehat{\mathcal{R}}_{\text{CE}}(\boldsymbol{W})$. Perform gradient descent for `num_iter` updates with a learning rate of `lrate`, initializing $\boldsymbol{W} = 0$ and returning $\boldsymbol{W}$ as output. Unlike previous problems, do *not* incorporate a bias term.

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

(c) [hw1code] Implement the `get_ntp_weights()` function in `hw1.py`. You are given as input the context size `n` and the embedding dimension `embedding_dim`. We will use a small subset of the TinyStories dataset Eldan and Li (2023) as our text sample, which you can extract using `hw1_utils.load_ntp_data()`. This function will return text data split into tokens `tokenized_data` (our tokenizer treats each word as a token), the list of all tokens in order of their id `sorted_words`, and the inverse mapping of token to id `word_to_idx`. You will then need to create the appropriate N-gram training data from `tokenized_data`. To do so, for every list of words in `tokenized_data` (which is a list of list of words), and create a training sample from every set of $n+1$ consecutive words (the first n words are the context, and the last word is the target). Given a list of w words, you should create exactly $w - n$ training samples (assuming $w \geq n$, of course). Use `hw1_utils.load_random_embeddings()` to get random feature embeddings for each word in the vocabulary. Run `cross_entropy(X,Y,C)` from part (b) taking `X`, `Y`, and vocabulary size `C` as input to obtain parameters $\boldsymbol{w}$. Use the default learning rate and number of iterations.

Library routines: `torch.stack.`

(d) [hw1code] Implement the `generate_text()` function in `hw1.py`. You are given as input the parameters $\boldsymbol{w}$ from the `get_ntp_weights()` function in part (c), the context size `n`, the number of additional tokens to generate `num_tokens`, the embedding dimension `embedding_dim`, and an initial context string `context`. When generating the next word, use a greedy policy, picking the word the model gives the highest probability. Return a string containing the initial context as well as all of the generated words, with each word separated by a space.

Library routines: `torch.argmax.`

(e) [hw1] Try generating at least 5 strings using different initial context strings using the `generate_text()` function in part (d) (use `n` $= 4$, `num_tokens` ≥ 20, and `embedding_dim` $= 10$), and include the results here. Do you notice anything unusual about the generated strings? Why do you think this happens?

Solution.

8. LLM Use and Other Sources.

[hw1] Please document, in detail, all your sources, including include LLMs, friends, internet resources, etc. For example:

1a. I asked my friend, then I found a different way to derive the same solution.

1b. ChatGPT 4o solved the problem in one shot, but then I rewrote it once one paper, and a few days later tried to re-derive an answer from scratch.

1c. I accidentally found this via a google search, and had trouble forgetting the answer I found, but still typed it from scratch without copy-paste.

1d. ...

⋮

6. I used my solution to problem 5 to write this answer.

Answer.

References

Ronen Eldan and Yuanzhi Li. Tinystories: How small can language models be and still speak coherent english?, 2023. URL https://arxiv.org/abs/2305.07759.