EECS545 WN24 Machine Learning
Homework #3

Due date: 11:55 PM, Tuesday March 5, 2024

Reminder: While you are encouraged to discuss problems in a small group (up to 5 people), you should write your solutions and code independently. Do not share your solution with anyone else in the class. If you worked in a group for the homework, please include the names of your collaborators in the homework submission. Your answers should be as concise and clear as possible. Please address all questions to `http://piazza.com/class#winter2024/eecs545` with a reference to the specific question in the subject line (e.g., Homework 3, Q2(c): foobar).

Submission Instruction: You should submit both **writeup** and **source code**. We may inspect your source code submission visually and rerun the code to check the results. Please be aware your points can be deducted if you don't follow the instructions listed below:

- Submit **writeup** to **Gradescope** (`https://www.gradescope.com/`)
  - Your writeup should contain your answers to **all** questions (typeset or hand-written), except for the implementation-only questions (marked with **(Autograder)**).
  - **For each subquestion, please select the associated pages from the pdf file in Gradescope. The graders are not required to look at pages other than the ones selected, and this may lead to some penalty when grading.**
  - Your writeup should be self-contained and self-explanatory. You should include the plots and figures in your writeup whenever asked, even when they are generated from the ipython notebook.
  - Please typeset (with LaTeX) or hand-write your solutions legibly. **Hand-writing that is difficult to read may lead to penalties**. If you prefer hand-writing, please use scanners or mobile scanning apps that provide shading corrections and projective transformations for better legibility; you **should *not*** just take photos and submit them without any image processing.
  - For all math derivations, you **should** provide detailed explanations as much as possible. If not, you may not be able to get a full credit if there are any mistakes or logical jumps.
- Submit **source code** to **Autograder** (`https://autograder.io/`)
  - Each ipython notebook file (i.e., `*.ipynb`) will walk you through the implementation task, producing the outputs and plots for *all* subproblems. You are required to write code on its matched python file (`*.py`). For instance, if you are working on `logistic_regression.ipynb`, you are required to write code on `logistic_regression.py`. Note that the outputs of your source code must match with your **writeup**.
  - We will read the data file in the `data` directory (i.e., `data/*.npy`) **from the same (current) working directory**: for example, `X_train = np.load('data/q1x.npy')`.
  - When you want to evaluate and test your implementation, please submit the `*.py` and `*.ipynb` files to Autograder for grading your implementations in the middle or after finish everything. You can partially test some of the files anytime, but please note that this also reduces your daily submission quota. You can submit your code up to **three** times per day.

- The autograder will give you information to help you troubleshoot your code. The following is the current list of error codes (will be updated accordingly if we encounter more common student implementation errors).
  - 0: Success
  - 1: The program exited with an unknown test failure.
  - 2: The program exited with an un-handled exception (usually a runtime error)
  - 3-5: Some other runtime error indicating an issue with the autograder code.
  - 40: NotImplementedError
  - 50: An uncategorized test failure different from the following error codes. Most likely indicates wrong outputs, but this may also include other assertion failures.
  - 51: There is an incorrect dtype on some output tensor.
  - 52: There is an incorrect shape on some output tensor.
  - 53: Some output tensors are not equal, or too different given the tolerance.yes
  - 54: Some output tensors are not equal with difference 1.0. Possibly due to rounding issues.
  - 55: The value of some input tensor was changed.
  - 60: The model's performance or converged loss is not good enough.
  - 61: The model might have diverged (e.g. encountered NaN or inf).
  - 62-63: The model has converged to a bad likelihood.
- Your program should run under an environment with following libraries:
  - Python 3.11 [1]
  - NumPy (for implementations of algorithms)
  - Matplotlib (for plots)
  - Cvxopt (for Q4) [2]
  - scikit-learn (for Q4)

  Please do not use any other library unless otherwise instructed (no third-party libraries other than numpy and matplotlib are allowed). You should be able to load the data and execute your code on Autograder with the environment described above, but in your local working environment you might need to install them via `pip install numpy matplotlib`.
- For this assignment, you will need to submit the following files:
  1. `soft_margin_svm.py`
  2. `soft_margin_svm.ipynb`
  3. `cvxopt_svm.py`
  4. `cvxopt_svm.ipynb`
  5. `two_layer_net.py`
  6. `two_layer_net.ipynb`

  Do not change the filename for the code and ipython notebook file because the Autograder may not find your submission. Please do not submit any other files except `*.py` and `*.ipynb` to Autograder.
- When you are done, please upload your work to Autograder (sign in with your UMich account). To receive the full credit, your code must run and terminate without error (i.e., with exit code 0) in the Autograder. Keep all the cell outputs in your notebook files (`*.ipynb`). We strongly recommend you run **Kernel → Restart Kernel and Run All Cells** (in the Jupyter Lab menu) before submitting your code to Autograder.

[1] We recommend using Miniconda (`https://docs.conda.io/en/latest/miniconda.html`). You can create `eecs545` environment with Python 3.11 as `conda create --name eecs545 python=3.11`. It is fine to use other python distributions and versions as long as they are supported, but we will run your program with Python 3.11 on Autograder.

[2] In some macOS environments (arm64 architecture), you might need to install cvxopt via a `conda install` command. Please have a look at the provided ipython notebook for more details.

Credits

Some problems were adopted from Stanford CS229 and Bishop PRML.

Change Log

- rev0 (2024/2/13): Initial Release

1 [21 + 3 points] Direct Construction of Valid Kernels

In class, we saw that by choosing a kernel $k(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^\top \phi(\mathbf{z})$, we can implicitly map data to a high dimensional space, and have the SVM algorithm work in that space. One way to generate kernels is to explicitly define the mapping ϕ to a higher dimensional space, and then work out the corresponding k.

However, in this question, we are interested in direct construction of kernels. Suppose we have a function $k(\mathbf{x}, \mathbf{z})$ that gives an appropriate similarity measure (similar to inner product) for our learning problem, and consider plugging k into a kernelized algorithm (like SVM) as the kernel function. In order for $k(\mathbf{x}, \mathbf{z})$ to be a valid kernel, it must correspond to an inner product in some higher dimensional space resulting from some feature mapping ϕ. In addition, Mercer's theorem states that $k(\mathbf{x}, \mathbf{z})$ is a (Mercer) kernel if and only if for any finite set $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)}\}$, the matrix $K \in \mathbb{R}^{N \times N}$ given by $K_{ij} = k\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right)$ is symmetric and positive semi-definite.

Now here comes the question: Let k_1, k_2 be kernels over $\mathbb{R}^D \times \mathbb{R}^D$, let $a \in \mathbb{R}^+$ be a positive real number, let $f : \mathbb{R}^D \to \mathbb{R}$ be a real-valued function, let $\phi : \mathbb{R}^D \to \mathbb{R}^M$ be a function mapping from $\mathbb{R}^D$ to $\mathbb{R}^M$, let k_3 be a kernel over $\mathbb{R}^M \times \mathbb{R}^M$, and let $p : \mathbb{R} \to \mathbb{R}$ be a polynomial with positive coefficients.

For each of the functions k below, state whether it is necessarily a kernel. If you think it is a kernel, please prove it; if you think it is not, please prove it or give a counterexample. You can use both definitions (inner prod of feature map or PSD) when proving the validity of the kernel. If you once proved a property in one sub-question (e.g., you proved Q1.(a)), then it is okay to use it in other questions, e.g., Q1.(f), by mentioning that you proved that property in Q1.(a); please make sure to provide the proof at least once. However, you are NOT allowed to use the 'results' from the 'Constructing kernels' slide without proving them yourself.

(a) **[2 points]** $k(\mathbf{x}, \mathbf{z}) = k_1(\mathbf{x}, \mathbf{z}) + k_2(\mathbf{x}, \mathbf{z})$

(b) **[2 points]** $k(\mathbf{x}, \mathbf{z}) = k_1(\mathbf{x}, \mathbf{z}) - k_2(\mathbf{x}, \mathbf{z})$

(c) **[2 points]** $k(\mathbf{x}, \mathbf{z}) = -ak_1(\mathbf{x}, \mathbf{z})$

(d) **[2 points]** $k(\mathbf{x}, \mathbf{z}) = f(\mathbf{x})f(\mathbf{z})$

(e) **[3 points]** $k(\mathbf{x}, \mathbf{z}) = k_3(\phi(\mathbf{x}), \phi(\mathbf{z}))$

(f) **[3 points]** $k(\mathbf{x}, \mathbf{z}) = p\left(k_1(\mathbf{x}, \mathbf{z})\right)$

(g) **[3 points]** $k(\mathbf{x}, \mathbf{z}) = k_1(\mathbf{x}, \mathbf{z})k_2(\mathbf{x}, \mathbf{z})$

(h) **[4 points]** For this sub-problem, you are not required to check if k is valid kernel. Instead, given a kernel $k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^\top \mathbf{z} + 1)^2$, find a feature map ϕ associated with $k(\mathbf{x}, \mathbf{z})$ such that $k(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^\top \phi(\mathbf{z})$. You may assume $D = 2$ for this subproblem.

(i) **[3 points, extra credit]** Prove that the Gaussian Kernel, $k(\mathbf{x}, \mathbf{z}) = \exp\left(-\|\mathbf{x} - \mathbf{z}\|^2 / 2\sigma^2\right)$ can be expressed as $k(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^\top \phi(\mathbf{z})$, where $\phi(\cdot)$ is an infinite-dimensional vector. Specifically, find the explicit closed form of an *infinite* dimensional feature vector ϕ. (Note: you cannot directly apply Mercer's theorem here as ϕ is an *infinite* dimensional vector)

[Hint 1: $\|\mathbf{x}-\mathbf{z}\|^2 = \mathbf{x}^\top \mathbf{x} + \mathbf{z}^\top \mathbf{z} - 2\mathbf{x}^\top \mathbf{z}$. *It might be useful to consider Power Series:* $\exp(x) = \sum_{n=0}^{\infty} \frac{1}{n!} x^n$*.]*

[Hint 2: You might find the formula in slide page 16 of Lecture 8 helpful.]

[Hint 3: Each element of $\phi(\mathbf{x})$ *can be written as an explicit formula over* $\mathbf{x}$ *without any limits or infinite summations.]*

2 [14 points] Implementing Soft Margin SVM by Optimizing Primal Objective

Support Vector Machines (SVM) is a discriminative model for classification. Although it is possible to develop SVMs that do K class classifications, we will restrict ourselves to binary classification in this question, where the class label is either $+1$ (positive) or -1 (negative). SVM is not a probabilistic algorithm. In other words, in its usual construction, it does not optimize a probability measure as a likelihood. SVM tries to find the "best" hyperplane that maximally separates the positive class from the negative class.

Recall that the objective function for maximizing the soft margin is equivalent to the following minimization problem:

$$\begin{aligned}
&\min_{\mathbf{w},b,\boldsymbol{\xi}} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi^{(i)} \\
&\text{subject to } y^{(i)}\left(\mathbf{w}^\top\phi(\mathbf{x}^{(i)}) + b\right) \geq 1 - \xi^{(i)},\ \forall i = 1,\ldots,N \\
&\xi^{(i)} \geq 0,\ \forall i = 1,\ldots,N
\end{aligned} \tag{1}$$

The above is known as the primal objective of SVM. Notice the two constraints on the slack variables $\xi^{(i)}$. It means that $\xi^{(i)}$ must satisfy both of those conditions, while minimizing the sum of $\xi^{(i)}$'s times C. The constrained minimization is equivalent to following minimization involving the *hinge loss* term:

$$\min_{\mathbf{w},b} E(\mathbf{w},b), \quad E(\mathbf{w},b) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\max\left(0, 1 - y^{(i)}\left(\mathbf{w}^\top\phi(\mathbf{x}^{(i)}) + b\right)\right) \tag{2}$$

You will be working with minimization of the above objective in this problem.

(a) **[5 points]** Find the derivatives of the loss function $E(\mathbf{w}, b)$ with respect to the parameters $\mathbf{w}, b$. Show that:

$$\nabla_{\mathbf{w}} E(\mathbf{w},b) = \mathbf{w} - C\sum_{i=1}^{N}\mathbb{I}\left[y^{(i)}\left(\mathbf{w}^\top\phi(\mathbf{x}^{(i)}) + b\right) < 1\right] y^{(i)}\phi(\mathbf{x}^{(i)}) \tag{3}$$

$$\frac{\partial}{\partial b} E(\mathbf{w},b) = -C\sum_{i=1}^{N}\mathbb{I}\left[y^{(i)}\left(\mathbf{w}^\top\phi(\mathbf{x}^{(i)}) + b\right) < 1\right] y^{(i)}, \tag{4}$$

where $\mathbb{I}[\cdot]$ is the indicator function. If $f(x) = \max(0, x)$, then assume that $\frac{\partial f}{\partial x} = \begin{cases} 1 & x > 0 \\ 0 & \text{otherwise} \end{cases}$

(b) **[7 points]** (Autograder) Implement the SVM algorithm using batch gradient descent, following part 1 of `soft_margin_svm.ipynb`. In previous assignments, you have implemented gradient descent while minimizing over one variable. Minimizing over two variables ($\mathbf{w}$, b) is not different. Both gradients are computed from current parameter values, and then parameters are simultaneously updated.[3]

Example pseudocode for optimizing $\mathbf{w}$ and b is given by Algorithm 1 below.

[3]Note: it is a common mistake to implement gradient descent over multiple parameters by updating the first parameter, then computing the derivative w.r.t second parameter using the updated value of the first parameter. In fact, updating one parameter then computing the gradient of the second parameter using the updated value of the first parameter, is a different optimization algorithm, known as Coordinate Descent.

Algorithm 1: SVM Batch Gradient Descent

```
$\mathbf{w}^* \leftarrow 0$
$b^* \leftarrow 0$
for $j = 1$ to NumEpochs do
    $\mathbf{w}_{grad} \leftarrow \nabla_{\mathbf{w}} E(\mathbf{w}^*, b^*)$
    $b_{grad} \leftarrow \frac{\partial}{\partial b} E(\mathbf{w}^*, b^*)$
    $\mathbf{w}^* \leftarrow \mathbf{w}^* - \eta(j)\mathbf{w}_{grad}$
    $b^* \leftarrow b^* - \eta(j) b_{grad}$
end
return $\mathbf{w}^*$, $b^*$
```

Throughout this question, we will set C in the previous question to $C = 5$, NumEpochs $= 100$ and the learning rate η as a constant. i.e., $\eta(j) = 1e^{-3}$).

(Remarks: In this problem, we only asked you to implement batch gradient descent for primal SVM, but it's quite straightforward to implement stochastic gradient descent in a very similar way (although you may need to decay the learning rate and set the initial learning rate properly). Unlike stochastic gradient descent, we don't need to decay the learning rate for convergence.)

(c) [**2 points**] Run gradient descent over the training data 5 times, once for each of the NumEpochs=[1, 3, 10, 30, 100]. For each run, please report your trained parameters $(\mathbf{w}, b)$ and the test classification accuracy in your **writeup**.

3 [20 points] Asymmetric Cost SVM

Consider applying an SVM to a supervised learning problem where the cost of a false positive (mistakenly predicting +1 when the label is -1) is different from the cost of a false negative (predicting -1 when the label is +1). The asymmetric cost SVM models these asymmetric costs by posing the following optimization problem:

$$\min_{\mathbf{w},b,\boldsymbol{\xi}} \frac{1}{2}\mathbf{w}^\top\mathbf{w} + C_0 \sum_{i:y^{(i)}=-1} \xi^{(i)} + C_1 \sum_{i:y^{(i)}=1} \xi^{(i)}$$

s.t.

$$y^{(i)}(\mathbf{w}^\top\phi(\mathbf{x}^{(i)}) + b) \geq 1 - \xi^{(i)}, \forall i = 1, \ldots, N$$
$$\xi^{(i)} \geq 0, \forall i = 1, \ldots, N$$

Here, C_0 is the cost of a false positive; C_1 is the cost of a false negative. (Both C_0 and C_1 are fixed, known, constants.)

(a) [**4 points**] We will find the dual optimization problem. First, write down the Lagrangian. Use $\alpha^{(i)}$ and $\mu^{(i)}$ to denote the Lagrange multipliers corresponding to the two constraints ($\alpha^{(i)}$ for the first constraint, and $\mu^{(i)}$ for the second constraint (slack variables)) in the primal optimization problem above.

$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\mu}) =$

(b) [**6 points**] Calculate the following derivatives with respect to the primal variables:

$\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\mu}) =$

$\frac{\partial\mathcal{L}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\mu})}{\partial b} =$

$\nabla_{\xi^{(i)}}\mathcal{L}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\mu}) =$

[Hint] You can define $C^{(i)}$ as $C^{(i)} = C_0\mathbb{I}[y^{(i)} = -1] + C_1\mathbb{I}[y^{(i)} = 1]$.

(c) [**10 points**] Find the dual optimization problem. You should write down the dual optimization problem in the following form. Try to simplify your answer as much as possible. In particular, to obtain full credit, the Lagrange multipliers $\mu^{(i)}$ should not appear in your simplified form of the dual.

$\max_{\boldsymbol{\alpha}} W(\boldsymbol{\alpha}) =$

s.t.

4 [18 points] SVMs with Convex Optimization

In this problem you will have practice training SVMs on toy data. You will learn how to use the popular `Scikit-Learn` library's SVM implementation and how to use the convex optimization library `CVXOPT` to train an SVM by directly solving the dual optimization problem.

(a) **[4 points]** Recall from lecture that the (kernelized) dual optimization problem of SVMs can be written as follows.

$$\begin{aligned} \underset{\boldsymbol{\alpha}}{\text{maximize}} \quad & \sum_{n=1}^{N} \alpha^{(n)} - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} \alpha^{(n)}\alpha^{(m)}y^{(n)}y^{(m)}k(\boldsymbol{x}^{(n)}, \boldsymbol{x}^{(m)}) \\ \text{subject to} \quad & 0 \leq \alpha^{(n)} \leq C \\ & \sum_{n=1}^{N} \alpha^{(n)}y^{(n)} = 0 \end{aligned} \tag{5}$$

where there are N training examples, $\boldsymbol{x}^{(i)} \in \mathbb{R}^D$, $y^{(i)} \in \{-1, 1\}$, k is a kernel function $k : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$, $\boldsymbol{\alpha} \in \mathbb{R}^N$, and $C \in \mathbb{R}$.

To solve this problem using `CVXOPT`, we will use the quadratic programming solver `cvxopt.solvers.qp`. Given matrices $\boldsymbol{P}, \boldsymbol{G}, \boldsymbol{A}$ and vectors $\boldsymbol{q}, \boldsymbol{h}, \boldsymbol{b}$, `CVXOPT` can solve the following optimization problem (a quadratic programming):

$$\begin{aligned} \underset{\boldsymbol{v}}{\text{minimize}} \quad & \frac{1}{2}\boldsymbol{v}^\top \boldsymbol{P}\boldsymbol{v} + \boldsymbol{q}^\top \boldsymbol{v} \\ \text{subject to} \quad & \boldsymbol{G}\boldsymbol{v} \preceq \boldsymbol{h} \\ & \boldsymbol{A}\boldsymbol{v} = \boldsymbol{b} \end{aligned} \tag{6}$$

Find values for matrices $\boldsymbol{P}, \boldsymbol{G}, \boldsymbol{A}$ and vectors $\boldsymbol{q}, \boldsymbol{h}, \boldsymbol{b}$ in terms of $\boldsymbol{x}^{(i)}, y^{(i)}, k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})$ and C such that the solution of the SVM dual problem ($\boldsymbol{\alpha}$ in Equation 5) is equal to the solution of the `QVXOPT` equation ($\boldsymbol{v}$ in Equation 6).

Hint 1. A maximization problem can be the same as a minimization when applying a sign change: $\min_x f(x) = \max_x -f(x)$.

Hint 2. $0 \leq \alpha^{(n)} \leq C$ *can be separated into two constraints:* $-\alpha^{(n)} \leq 0$ *and* $\alpha^{(n)} \leq C$.

(b) **[10 points]** (Autograder) Use your derivation above to complete the code in the `svm.py` file.

(c) **[4 points]** In the notebook, you will implement drawing a decision boundary on a simple dataset and identify support vectors from the solutions to the optimization problems. Include the test performance on each dataset and the generated visualizations of the cvxopt SVMs in your submission writeup (generated in `svm.ipynb`).

5 [27 points] Neural Network Layer Implementation

In this problem, you will get the opportunity to implement various neural network layers. $\mathbf{X}, \mathbf{Y}$ will represent the input and the output of the layers, respectively. For this problem, we use the row-major notation here (i.e. each row of $\mathbf{X}$ and $\mathbf{Y}$ corresponds to one data sample) to be compatible with the multi-dimensional array structure of NumPy. Furthermore, $\mathcal{L}$ is the scalar valued loss function of $\mathbf{Y}$.

For the Fully-Connected Layer and ReLU, you should include your derivation of the gradient in your written solution.

Important Note: In this question, any indices involved (i, j, etc.) in the mathematical notation start from 1, and not 0. In your code, however, you should use 0-based indexing (standard NumPy notation).

(a) **[8 points] Fully-Connected Layer**
In this question, you will be deriving the **gradient** of a fully-connected layer. For this problem, consider $\mathbf{X} \in \mathbb{R}^{N \times D_{\text{in}}}$, where N is the number of samples in a batch. Additionally, consider a dense layer with weight parameters of the form of $\mathbf{W} \in \mathbb{R}^{D_{\text{in}} \times D_{\text{out}}}$ and bias parameters of the form of $\mathbf{b} \in \mathbb{R}^{1 \times D_{\text{out}}}$.

As we saw in the lecture, we can compute the output of the layer through a simple matrix multiply operation. Concretely, this can be seen as $\mathbf{Y} = \mathbf{XW} + \mathbf{B}$. In this equation, we denote the output matrix as $\mathbf{Y} \in \mathbb{R}^{N \times D_{\text{out}}}$ and the bias matrix as $\mathbf{B} \in \mathbb{R}^{N \times D_{\text{out}}}$ where each row of $\mathbf{B}$ is the vector $\mathbf{b}$. (Please note that we are using row-vector notation here to make it compatible with NumPy/PyTorch, but the lecture slides used column-vector notation, which is standard notation for most ML methods. We hope that the distinction is clear from the context.)

Please note, that the matrix multiply operation stated above is generally useful to compute batch outputs (i.e. simultaneously computing the output of N samples in the batch). However, for the purposes of computing the gradient, you might first want to consider the single-sample output operation of this fully-connected layer, which can be stated in row-major notation as $\mathbf{y}^{(n)} = \mathbf{x}^{(n)}\mathbf{W} + \mathbf{b}$ where $\mathbf{y}^{(n)} \in \mathbb{R}^{1 \times D_{out}}$ and $\mathbf{x}^{(n)} \in \mathbb{R}^{1 \times D_{in}}$ for $1 \le n \le N$. Here, the index "(n)" in the superscript denotes n-th example, not the layer index, and $X_i^{(n)} = X_{n,i}$ denotes i-th dimension of n-th example $\mathbf{x}^{(n)}$.

Note: it might be fruitful for you to consider this expression as a summation and then calculate the gradient for individual parameters for a single-example case. After this, you can try to extend it to a vector/matrix format for the involved parameters.

Now, **compute the partial derivatives (in matrix form)** $\frac{\partial \mathcal{L}}{\partial \mathbf{W}} \triangleq \nabla_{\mathbf{W}}\mathcal{L} \in \mathbb{R}^{D_{\text{in}} \times D_{\text{out}}}$, $\frac{\partial \mathcal{L}}{\partial b} \triangleq \nabla_{b}\mathcal{L} \in \mathbb{R}^{1 \times D_{\text{out}}}$, $\frac{\partial \mathcal{L}}{\partial \mathbf{X}} \triangleq \nabla_{\mathbf{X}}\mathcal{L} \in \mathbb{R}^{N \times D_{\text{in}}}$ **in terms of** $\frac{\partial \mathcal{L}}{\partial \mathbf{Y}} \triangleq \nabla_{\mathbf{Y}}\mathcal{L} \in \mathbb{R}^{N \times D_{\text{out}}}$.[4] For technical correctness, you might want to start with writing the gradient with non-vectorized form involving $\frac{\partial \mathcal{L}}{\partial Y_j^{(n)}} \in \mathbb{R}$, where $\frac{\partial \mathcal{L}}{\partial Y_j^{(n)}}$ is the gradient with respect to j-th element of n-th sample in $\mathbf{Y}$ with $1 \le n \le N$ and $1 \le j \le D_{\text{out}}$.

Hint for $\frac{\partial \mathcal{L}}{\partial \mathbf{W}}$
Please note that we use a "matrix/vector" notation $\frac{\partial \mathcal{L}}{\partial \mathbf{W}}$ to denote a matrix in $\mathbb{R}^{D_{\text{in}} \times D_{\text{out}}}$ where the element in i-th row and j-th column is $\frac{\partial \mathcal{L}}{\partial W_{i,j}}$ and $W_{i,j}$ is the i-th row, j-th column element of $\mathbf{W}$ with $1 \le i \le D_{\text{in}}$ and $1 \le j \le D_{\text{out}}$. Here, you may want to calculate the gradient $\frac{\partial \mathcal{L}}{\partial W_{i,j}}$ using the formula $\frac{\partial \mathcal{L}}{\partial W_{i,j}} = \sum_{n=1}^{N} \frac{\partial \mathcal{L}}{\partial Y_j^{(n)}} \frac{\partial Y_j^{(n)}}{\partial W_{i,j}}$.

Hint for $\frac{\partial \mathcal{L}}{\partial \mathbf{b}}$
Please note that $\frac{\partial \mathcal{L}}{\partial \mathbf{b}}$ is a vector in $\mathbb{R}^{1 \times D_{\text{out}}}$. You may want to start by using the formula $\frac{\partial \mathcal{L}}{\partial b_j} =$

[4] $\triangleq$ means 'equal to by definition'

$\sum_{n=1}^{N}\sum_{m=1}^{D_{\text{out}}} \frac{\partial \mathcal{L}}{\partial Y_m^{(n)}} \frac{\partial Y_m^{(n)}}{\partial b_j}$ and then move on to derive $\frac{\partial \mathcal{L}}{\partial \mathbf{b}}$.

Hint for $\frac{\partial \mathcal{L}}{\partial \mathbf{X}}$
Please note that $\frac{\partial \mathcal{L}}{\partial \mathbf{X}}$ is a matrix in $\mathbb{R}^{N \times D_{in}}$. In order to derive this gradient, you can start by using the formula $\frac{\partial \mathcal{L}}{\partial X_i^{(n)}} = \sum_{n'=1}^{N}\sum_{m=1}^{D_{\text{out}}} \frac{\partial \mathcal{L}}{\partial Y_m^{(n')}} \frac{\partial Y_m^{(n')}}{\partial X_i^{(n)}}$ Following this, you can say that $\frac{\partial \mathcal{L}}{\partial X_i^{(n)}} = \sum_{m=1}^{D_{\text{out}}} \frac{\partial \mathcal{L}}{\partial Y_m^{(n)}} \frac{\partial Y_m^{(n)}}{\partial X_i^{(n)}}$ because the n-th sample in X is only related with the n-th sample in $\mathbf{Y}$ and $\frac{Y_m^{(n')}}{\partial X_i^{(n)}}$=0 for all $n' \neq n$.

(b) **[3 points] ReLU**
Let $\mathbf{X} \in \mathbb{R}^{N \times D}$ be a 2-D array and $\mathbf{Y} = \text{ReLU}(\mathbf{X})$. In this case, ReLU is applied to $\mathbf{X}$ in an element-wise fashion. For an element $x \in \mathbf{X}$, the corresponding output is $y = \text{ReLU}(x) = \max(0, x)$. You can observe that $\mathbf{Y}$ will have the same shape as $\mathbf{X}$. **Express $\frac{\partial \mathcal{L}}{\partial \mathbf{X}}$ in terms of $\frac{\partial \mathcal{L}}{\partial \mathbf{Y}}$**, where $\frac{\partial \mathcal{L}}{\partial \mathbf{X}}$ has the same shape as $\mathbf{X}$.

Hint: you may need to use the element-wise product to express $\frac{\partial \mathcal{L}}{\partial \mathbf{X}}$. Please feel free to introduce the indicator function $\mathbb{I}[\cdot]$.

(c) **[14 points]** (Autograder) Now, we will implement the corresponding forward and backward passes in `two_layer_net.py`. More importantly, please ensure that you use vectorization to make your layer implementations as efficient as possible. You should use `two_layer_net.ipynb` to help you in checking the correctness of the your implementation. Note that for each layer, the ipython notebook just checks the result for *one specific example*. Passing these tests does not necessarily mean that your implementation is 100% correct. Once you implement the layers properly, please implement two fully-connected layer based classifier with a Softmax loss in `two_layer_net.py` and test it on MNIST dataset, following the guide in `two_layer_net.ipynb`. All your implementations should go to `two_layer_net.py`.

(d) **[1 points]** Please report the training loss with train/test accuracy plot of MNIST dataset, stored as `two_layer_net.png` from `two_layer_net.ipynb`, in your **writeup**.

(e) **[1 points]** Please report the accuracy obtained on the test set of MNIST dataset in your **writeup**.