by Peter Tang and Ximeng Sun. Modified from the version by Ruizhao Zhu

CS/EC523 Deep Learning

Problem Set 1

Preamble

To run and solve this assignment, you need an interface to edit and run ipython notebooks (`.ipynb` files). The easiest way to complete this assignment is to use Google Colab. You can just copy the assignment notebook to your google drive and open it, edit it and run it on Google Colab (https://colab.research.google.com/). All libraries you need are pre-installed on Colab.

Submission instructions: please upload your completed solution files (having run all code cells and rendered all markdown/Latex) to Gradescope by the due date (see Schedule for due dates and late policy). Please upload both the notebook and the PDF (File->Download as->PDF)

Gradescope link: https://www.gradescope.com/courses/363264 (https://www.gradescope.com/courses/363264)

Entry Code: ZR23VJ

Local installation

The alternative is to have a local installation, although we do not recommend it. If you are working on Google Colab, feel free to skip to the next section "More instructions". We recommend using virtual environments for all your installations. Following is one way to set up a working environment on your local machine for this assignment, using [Anaconda (https://www.anaconda.com/distribution/)](https://www.anaconda.com/distribution/):

- Download and install Anaconda following the instructions [here (https://docs.anaconda.com/anaconda/install/)](https://docs.anaconda.com/anaconda/install/)
- Create a conda environment using `conda create --name dl_env python=3` (You can change the name of the environment instead of calling it `dl_env`)
- Now activate the environment using : `conda activate dl_env`
- Install jupyter lab, which is the [jupyter project's (https://jupyter.org/index.html)](https://jupyter.org/index.html) latest notebook interface : `pip install jupyterlab`. You can also use the classic jupyter notebooks and there isn't any difference except the interface.
- Install other necessary libraries. For this assignment you need `numpy`, `scipy`, [`pytorch` (https://pytorch.org/get-started/locally/)](https://pytorch.org/get-started/locally/) and `matplotlib`, all of which can be installed using : `pip install <lib_name>`. Doing this in the environment, would install these libraries for `dl_env`. You can also use `conda install`.
- Now download the assignment notebook in a local directory and launching `jupyter lab` in the same directory should open a jupyter lab session in your default browser, where you can open and edit the ipython notebook.
- For deactivating the environment when you are done with it, use : `conda deactivate`.

For users running a Jupyter server on a remote machine :

- Launch Jupyter lab on the remote server (in the directory with the homework ipynb file) using : `jupyter lab --no-browser --ip=0.0.0.0`
- To access the jupyter lab interface on your local browser, you need to set up ssh port forwarding. This can be done by running : `ssh -N -f -L localhost:8888:localhost:8888 <remoteuser>@<remotehost>`. You can now open `localhost:8888` on your local browser to access jupyter lab. This assumes you are running jupyter lab on its default port 8888 on the server.
- Check "Making life easy" section at the end of [this post (https://ljvmiranda921.github.io/notebook/2018/01/31/running-a-jupyter-notebook/)](https://ljvmiranda921.github.io/notebook/2018/01/31/running-a-jupyter-notebook/) to find how to add functions to your bash run config to do this more easily each time. The post mentions functions for jupyter notebook, but just replace those with jupyter lab if you are using that interface.

The above instructions specify one way of working on the assignment. You can use other virtual environments/ipython notebook interfaces etc. (**not recommended**).

More instructions

If you are new to Python or its scientific library, Numpy, there are some nice tutorials [here (https://www.learnpython.org/)](https://www.learnpython.org/) and [here (http://www.scipy-lectures.org/)](http://www.scipy-lectures.org/).

In an ipython notebook, to run code in a cell or to render [Markdown (https://en.wikipedia.org/wiki/Markdown)](https://en.wikipedia.org/wiki/Markdown)+[LaTeX (https://en.wikipedia.org/wiki/LaTeX)](https://en.wikipedia.org/wiki/LaTeX) press `Ctrl+Enter` or `[>|]` (like "play") button above. To edit any code or text cell (double) click on its content. To change cell type, choose "Markdown" or "Code" in the drop-down menu above.

To enter your solutions for the written questions, put down your derivations into the corresponding cells below using LaTeX. Show all steps when proving statements. If you are not familiar with LaTeX, you should look at some tutorials and at the examples listed below between $..$. We will not accept handwritten solutions.

Put your solutions into boxes marked with **`[double click here to add a solution]`** and press Ctrl+Enter to render text. (Double) click on a cell to edit or to see its source code. You can add cells via **`+`** sign at the top left corner.

Note: `Vector` stands for `column vector` below.

Problem 1: Review of Probability and Statistics [15 points]

Q1.1: X and Y are random variables. The expectaion of X is $E(X)$ and the variance of X is $Var(X)$. The expectaion of Y is $E(Y)$ and the variance of X is $Var(Y)$. Prove the following:

(a) $E(aX + bY) = aE(X) + bE(Y)$

(b) $Var(aX + bY) = a^2 Var(X) + b^2 Var(Y) + 2abCov(X, Y)$, where $Cov(X, Y)$ is covariance of X and Y.

[double click here to add a solution]

(a)

$$E(aX + bY) = \Sigma_{X,Y}[(aX + bY) \times P_{X,Y}(X + Y)]$$
$$E(aX + bY) = \Sigma_{X,Y}\, aX \times P_{X,Y}(X + Y) + \Sigma_{Y,X}\, bY \times P_{X,Y}(X + Y)$$
$$E(aX + bY) = a\Sigma_X X \times P_X(X) + b\Sigma_Y Y \times P_Y(Y)$$
$$E(aX + bY) = aE(X) + bE(Y)$$

(b)

$$Cov(X, Y) = E(X, Y) - (E(X) \times E(Y))$$
$$2abCov(X, Y) = 2ab[E(X, Y) - (E(X) \times E(Y))]$$
$$Var(aX + bY) = a^2 Var(X) + b^2 Var(Y) + 2Cov(aX + bY)$$
$$Var(aX + bY) = E((aX + bY)^2) - (E(aX + bY))^2$$
$$= E(a^2X^2 + 2abXY + b^2Y^2) - (E(aX) + E(bY))^2$$
$$= a^2E(X^2) + 2abE(X, Y) + b^2E(Y^2) - (E(aX)^2 + 2E(aXbY) + E(bY)^2)$$
$$= a^2E(X^2) + 2abE(X, Y) + b^2E(Y^2) - a^2E(X)^2 - 2abE(X, Y) - b^2E(Y)^2$$
$$= a^2E(X^2) - a^2E(X)^2 + b^2E(Y^2) - b^2E(Y)^2 + 2abE(X, Y) - 2abE(X, Y)$$
$$= a^2(E(X^2) - E(X)^2) + b^2(E(Y^2) - E(Y)^2) + 2ab(E(X, Y) - E(X, Y))$$
$$= a^2Var(X) + b^2Var(Y) + 2abCov(X + Y)$$

Q1.2 Suppose $X_1, \ldots, X_N$ have mean μ and variance σ^2 and are independent. Let $A = (X_1 + .. + X_N)/N$ be the empirical mean. Show that with probability at least 0.2, $|A - \mu| < \frac{\sqrt{5}\sigma}{2\sqrt{N}}$

[double click here to add a solution]

Q1.3: Assume that we are given n iid samples $(x_1, \ldots, x_n)$ from each $P(x \mid \theta)$ given below. Compute the maximum likelihood estimates (MLEs) for the parameter θ (α, β) of the given distributions.

Q1.3.1 $P(x \mid \theta) = \theta e^{-\theta x^2}$ for $x \geq 0$

Q1.3.2 $P(x \mid \theta) = \frac{1}{1-\theta}$ for $\theta \leq x \leq 1$

Q1.3.3 $P(x \mid \alpha, \beta) = \frac{1}{\pi\alpha[1+(\frac{x-\beta}{\alpha})^2]}$

(If x is not in the support of the distribution defined by inequalities, the probability of it is 0.)

Hint for Q1.3.3: Don't solve for α or β, just simply the equation as much as you can.

[double click here to add a solution]

Q1.4 (Naive Bayes Maximum Likelihood) Consider binary dataset S with observations in the form $\left\{\left(x_j^1, \ldots, x_j^n\right), y_j\right)\}$. Each data $\mathbf{x_j}$ is an n-dimensional vector. Define $c(y)$ as a function that counts the number of observations such that the label is y. S is the set of all the training data.

$$c(y) = \sum_{(x_j, y_j) \in S} [y_j = y]$$

Define $c(i, y)$ as a function that counts the number of observations such that the label is y and $x^i = 1$

$$c(i, y) = \sum_{(x_j, y_j) \in S} [y_j = y, x_j^i = 1]$$

Define b as $P(Y = 1)$, and b^{iy} as $P\left(X^i = 1 \mid Y = y\right)$. Prove that the following estimators are MLE for these parameters:

$$\hat{b}_{MLE} = \frac{c(1)}{|S|} \quad \text{and} \quad \widehat{b^{iy}}_{MLE} = \frac{c(i, y)}{c(y)}$$

[double click here to add a solution]

Problem 2: Matrix Derivatives [15 points]

Multivariate Gaussian

Assume that our data is distributed according to a d dimensional [multivariate Gaussian (https://en.wikipedia.org/wiki/Multivariate_normal_distribution#Likelihood_function)](https://en.wikipedia.org/wiki/Multivariate_normal_distribution#Likelihood_function) with $\bar{\mu}$ mean and Σ covariance matrix:

$$(\mathbf{x}_1, \ldots, \mathbf{x}_n) \sim \mathcal{N}(\bar{\mu}, \Sigma).$$

Q2.1: Using rules of [matrix derivatives (http://www2.imm.dtu.dk/pubdb/views/edoc_download.php/3274/pdf/imm3274.pdf)](http://www2.imm.dtu.dk/pubdb/views/edoc_download.php/3274/pdf/imm3274.pdf) Equation [57, 59], derive $\frac{\partial \mathcal{L}(\theta)}{\partial \Sigma}$ in matrix form and set it to zero to find Σ_{ML}. Assume each x_i is drawn independently.

[double click here to add a solution]

$$\frac{\partial \mathcal{L}(\theta)}{\partial \Sigma} = \frac{\partial}{\partial \Sigma}[\frac{1}{2}(\sigma(wx + b) - t)^2]$$
$$0 = \frac{\partial}{\partial \Sigma}[\frac{1}{2}(\sigma(wx + b) - t)^2]$$

$$ 0 =

Q2.2: Multi-target Linear Regression

- we have $X \in \mathbf{R}^{n \times d}$ is a constant data matrix
- θ is a $d \times m$-dimensional **weight matrix**
- $\varepsilon_{ij} \sim \mathcal{N}(0, \sigma_\epsilon^2)$ is a normal noise ($i \in [0; n], j \in [0; m]$)
- and we observe a matrix $Y = X\theta + \varepsilon \in \mathbf{R}^{n \times m}$

$$\varepsilon = Y - X\theta \sim \mathcal{N}(0, \sigma_\epsilon^2 I)$$
$$\mathcal{L}(\theta) = \log P(Y \mid X, \theta) = \log \mathcal{N}(Y - X\theta \mid 0, \sigma_\epsilon^2 I)$$
$$\theta_{MLE} = \arg\max_\theta \mathcal{L}(\theta) = \arg\min_\theta \operatorname{loss}(\theta) = \arg\min_\theta \left(||Y - X\theta||_F^2 \right)$$

Assume $\theta \sim \mathcal{N}(0, \sigma_\theta^2 I)$, which essentially means that "weight components should not be too far from zero". Where I stands for an identity matrix.

Using rules of [matrix derivatives](http://www2.imm.dtu.dk/pubdb/views/edoc_download.php/3274/pdf/imm3274.pdf) (http://www2.imm.dtu.dk/pubdb/views/edoc_download.php/3274/pdf/imm3274.pdf) Equation [137, 132], show for an MLE loss: $\operatorname{loss}(\theta) = ||Y - X\theta||_F^2$ that:

Q2.2.1: derive $\frac{\partial \operatorname{loss}(\theta)}{\partial \theta} = -2X^T(Y - X\theta)$

Q2.2.2: derive $\theta_{MLE} = (X^T X)^{-1} X^T Y$

Hint: In our case [see Matrix Cookbook, eq. 137], $g(U) = ||U||_F^2$ - squared Frobenius norm and $U(\theta) = f(\theta) = Y - X\theta$ - linear mapping.

Note: That is a multi-target problem, therfore θ is a matrix, so you have to take the derivative wrt matrix.

Q2.2.3 To make use of prior information, one might want to find [maximum a posteriori estimation](https://en.wikipedia.org/wiki/Maximum_a_posteriori_estimation) (https://en.wikipedia.org/wiki/Maximum_a_posteriori_estimation) (MAP). In this problem, the MAP is defined as $\theta_{MAP} = \arg\max_\theta L_{MAP}(\theta) = \arg\max_\theta P(\theta | X, Y)$. The priori of θ is $P(\theta)$. Show $\mathcal{L}_{MAP}(\theta)$ can be defined as $\mathcal{L}_{MLE}(\theta) + \log P(\theta)$.

Q2.2.4 Assume the prior $\theta \sim \mathcal{N}(0, \sigma_\theta^2 I)$, which essentially means that "weight vector components should not be too far from zero". Show MAP with gaussian prior is L2 regularization.

Q2.2.5 Now assume that a vectorized θ follows a [Laplace distribution](https://en.wikipedia.org/wiki/Laplace_distribution) (https://en.wikipedia.org/wiki/Laplace_distribution) i.e: $\theta_i \sim Laplace(0, b), \forall i$ Show MAP with laplace prior is L1 regularization.

`[double click here to add a solution]`

Problem 3: Ridge and Lasso Regression [15 points]

In this problem, we want to solve the regression problem with two different machine learning methods: Ridge Regression and Lasso Regression.

- We have the constant data matrix $X \in \mathbf{R}^{n \times d}$. Each row $\mathbf{x_i}$ is a d-dimensional data vector.
- We have the vector $\mathbf{w} \in \mathbf{R}^d$ and noise vector $\epsilon \in \mathbf{R}^n$.
- We observe $y_i \in \mathbf{R}$ as the output for each row in the input data matrix $\mathbf{x_i}$: $y_i = \mathbf{w}^T \mathbf{x_i} + \epsilon_i$

Given the model definition above, please derive the following:

Q3.1: Ridge Regression can be formulated as:

$$\mathbf{w}^* = \text{argmin}_{\mathbf{w} \in \mathbf{R}^d} \sum_i (y_i - \mathbf{w}^T \mathbf{x_i})^2 + \lambda_r ||\mathbf{w}||_2^2,$$

where $\lambda_r > 0$ is the hyperparameter to control the L2 regularization of $\mathbf{w}$. The loss function for Ridge Regression is $J(\mathbf{w}, \lambda_r) = \sum_i (y_i - \mathbf{w}^T \mathbf{x_i})^2 + \lambda_r ||\mathbf{w}||_2^2$ Derive the closed form for the optimal $\mathbf{w}^*$ in terms of $\mathbf{X}$, $\mathbf{y}$ and λ_r.

[double click here to add a solution]

$$\mathcal{L} = ||y - Xw^*||_2^2 + \lambda ||w^*||_2^2$$
$$\frac{d\mathcal{L}}{dw^*} = -2X^T(y - Xw^*) + 2\lambda w^*$$
$$0 = -2X^T y + 2X^T X w^* + 2\lambda w^*$$
$$X^T y = (X^T X + \lambda I) w^*$$
$$w^* = (X^T X + \lambda I)^{-1} X^T y$$

Q3.2: We can also use L1 regularization of $\mathbf{w}$ instead of L2 regularization in the Ridge Regression, which formulates the Lasso Regression:

$$\mathbf{w}^* \in \text{argmin}_{\mathbf{w} \in \mathbf{R}^d} \sum_i (y_i - \mathbf{w}^T \mathbf{x_i})^2 + \lambda_l ||\mathbf{w}||_1,$$

where $\lambda_l > 0$ is the hyperparameter to control the L1 regularization of $\mathbf{w}$. The loss function for Lasso Regression is $J(\mathbf{w}, \lambda_l) = \sum_i (y_i - \mathbf{w}^T \mathbf{x_i})^2 + \lambda_l ||\mathbf{w}||_1$. Show that Lasso Regression is not differentiable at some points.

[double click here to add a solution]

Q3.3: In this question, we will show the maximum value for λ_l is $2||\mathbf{X}^T\mathbf{y}||_\infty$ in Lasso Regression, i.e. for any $\lambda_l \geq 2||\mathbf{X}^T\mathbf{y}||_\infty$, the optimal $\mathbf{w}$ is always a 0-vector. We will gradually show this in three steps.

Q3.3.1: The one-sided directional derivative of a function $f(x)$ at the direction $\mathbf{u} \in \mathbf{R}^d$ is defined as follows.

$$f'(x;\mathbf{u}) = \lim_{h\to 0^+} \frac{f(x+h\mathbf{u}) - f(x)}{h},$$

where $\mathbf{u}$ is a unit vector. Please compute the one-sided directional derivative at $J(\mathbf{0}, \lambda_l)$ at direction $\mathbf{u}$. The final result (i.e. $J'(\mathbf{0}, \lambda_l; \mathbf{u})$) should be in terms of $\mathbf{X}$ (matrix), $\mathbf{y}$ (vector), λ_l and $\mathbf{u}$ (vector).

[double click here to add a solution]

Q3.3.2 Show that for any $\mathbf{u} \neq \mathbf{0}$, we have $J'(\mathbf{0}, \lambda_l; \mathbf{u}) \geq 0$ if and only if λ_l is greater than or equal to some constant C, which depends on $\mathbf{X}$ (matrix), $\mathbf{y}$ (vector) and $\mathbf{u}$ (vector). Please give an explicit expression for C.

[double click here to add a solution]

Q3.3.3: Due to the convexity of Lasso Regression, $\mathbf{w}^*$ is a minimzer of $J(\mathbf{w}, \lambda_l)$ if and only if the directional derivative $J'(\mathbf{w}^*, \lambda_l; \mathbf{u}) \geq 0$ for all $\mathbf{u} \neq 0$. Show that $\mathbf{w} = 0$ is the minimizer of $J(\mathbf{w}, \lambda_l)$ if and only if $\lambda_l \geq 2||\mathbf{X}^T\mathbf{y}||_\infty$.

[double click here to add a solution]

Problem 4: Coding Ridge and Lasso Regression [15 points]

In this problem, we will solve Ridge and Lasso Regression with Stochastic Gradient Descent (SGD) with the synthetic data. We will compute the (sub)gradient with the training data manually and update the weight vector $\mathbf{w}$. With the validation data, we tune the hyperparameters λ_r and λ_l. Finally we compare the learned weights learned by Ridge Regression and Lasso Regression.

Q4.1 Compute the gradient $\frac{\partial J(\mathbf{w},\lambda_r)}{\partial \mathbf{w}}$ of Ridge Regression. Also, compute $\frac{\partial J(\mathbf{w},\lambda_l)}{\partial \mathbf{w}}$ at all differentiable points and the subgradient at non-differentiable points in Lasso Regression.

[double click here to add a solution]

Q4.2 Implement the Ridge and Lasso Regression. We provide the synthetic data in `lasso_data_ours.pickle` in which $\mathbf{y} = \mathbf{X}\mathbf{w} + noise$. Please implement the ridge and lasso regression to learn the weight $\mathbf{w}$ with **Stochastic** Gradient Descent. For the differentiable $\mathbf{w}$, we use the gradient to update $\mathbf{w}$ and for non-differentiable $\mathbf{w}$, we use the subgradient instead. You need to fill in the missing part (starting with `## -- ! code required` comment) in `Ridge_loss`, `Ridge_grad`, `Lasso_loss`, `Lasso_grad` and `training`. Note: in this homework, you are required to use the gradient of the loss function computed manually (in Q5.1) instead of using the `autograd` in any deep learning library. Visualize the learned $\mathbf{w}$ and find out the difference between $\mathbf{w}$s learned by Ridge and Lasso Regression.

In [3]:

```
import pickle
import numpy as np
from copy import deepcopy
import matplotlib.pyplot as plt

# Change SALIENT to False to print the intermediate losses during trai
ning
SALIENT = True

np.random.seed(42)

class AverageMeter(object):
    def __init__(self):
        self.reset()

    def reset(self):
        self.val = 0
        self.avg = 0
        self.sum = 0
        self.count = 0

    def update(self, val, n=1):
        self.val = val
        self.sum += val * n
        self.count += n
        self.avg = self.sum / self.count


def permute_data(x, y):
    if y.ndim == 1:
        y = np.expand_dims(y, axis=-1)
    cat_xy = np.concatenate((x, y), axis=-1)
    cat_xy = np.random.permutation(cat_xy)
    return cat_xy[:, :-1], cat_xy[:, -1]


def l2_norm_square(x):
    return np.sum(np.power(x, 2))


def mse(X, y, w):
```

```python
    return l2_norm_square(np.matmul(X, w) - y)


def Ridge_loss(X, y, w, LAMBDA):
    """
    X: n x d matrix
    y: column vector of size (n,)
    w: column vector of size (d, )
    LAMBDA: a float scalar
    Return: a float scalar showing ridge regression loss with the give
n X, y, w, LAMBDA
    """
    #return np.linalg.norm(y - X.dot(w)) ** 2 / X.shape[0] + LAMBDA *
np.linalg.norm(w) ** 2
    #return np.sum(np.square(y - X.dot(w))) + LAMBDA * np.sum(np.squar
e(w))
    #return np.linalg.norm(y - X.dot(w))**2 + LAMBDA * np.linalg.norm(
w)**2
    #loss = np.linalg.norm(X.dot(w) - y) ** 2 + LAMBDA * np.linalg.nor
m(w) ** 2
    return np.linalg.norm(X.dot(w) - y)**2 + LAMBDA * np.linalg.norm(w
)**2
    #return (np.transpose(y - X @ w)) @ (y - X @ w) + LAMBDA * np.tran
spose(w) @ w


def Ridge_grad(X, y, w, LAMBDA):
    """
    X: n x d matrix
    y: column vector of size (n,)
    w: column vector of size (d, )
    LAMBDA: a float scalar
    Return: a column vector of size (d,) showing gradient of ridge reg
ression loss with the given X, y, w, LAMBDA

    """
    #return 2 * (X.T.dot(X.dot(w) - y) + LAMBDA * w)
    #return 2 * (X.T @ X @ w - X.T @ y + LAMBDA * np.sign(w))
    #return 2 * (X.T @ X @ w - X.T @ y + LAMBDA * np.sign(w))
    return 2 * X.T.dot(X.dot(w) - y) + 2 * LAMBDA * w
    #return -2 * np.transpose(X) @ Y + 2 * np.transpose(X) @ X @ w + 2
* LAMBDA * w


def Lasso_loss(X, y, w, LAMBDA):
    """
    X: n x d matrix
    y: column vector of size (n,)
    w: column vector of size (d, )
    LAMBDA: a float scalar
    Return: a float scalar showing Lasso regression loss with the give
n X, y, w, LAMBDA
    """
```

```
    #return np.sum((y - np.dot(X, w)) ** 2) + LAMBDA * np.sum(np.abs(w
))
    #return np.sum(np.abs(X.dot(w) - y)) + LAMBDA * np.sum(np.abs(w))
    #return np.linalg.norm(X.dot(w) - y)**2 + LAMBDA * np.linalg.norm(
w, ord=1)
    #return np.sum(np.abs(X.dot(w) - y)) + LAMBDA * np.sum(np.abs(w))
    return np.linalg.norm(X.dot(w) - y)**2 + LAMBDA * np.sum(np.abs(w)
)
    #return (np.transpose(y - X@w))@(y - X@w) + LAMBDA * np.linalg.nor
m(w,1)


def Lasso_grad(X, y, w, LAMBDA):
    """
    X: n x d matrix
    y: column vector of size (n,)
    w: column vector of size (d, )
    LAMBDA: a float scalar
    Return: a column vector of size (d,) showing gradient of lasso reg
ression loss with the given X, y, w, LAMBDA
    """
    #return 2*(X.T@X@w-X.T@y+LAMBDA*np.sign(w))
    #return 2 * (X.T @ X @ w - X.T @ y + LAMBDA * np.sign(w))
    #return (np.dot(X.T, np.dot(X, w) - y) + LAMBDA * np.sign(w))
    #return X.T.dot(X.dot(w) - y) + LAMBDA * np.sign(w)
    return 2 * X.T.dot(X.dot(w) - y) + LAMBDA * np.sign(w)
    #return -2 * np.transpose(X)@y + 2 * np.transpose(X)@X@w + LAMBDA
* np.sign(2)


def training(X_train, y_train, X_validation, y_validation, w, LAMBDA,
STEP_SIZE, method='Ridge'):
    BATCH_SIZE = 16
    NUM_TRAIN = len(X_train)
    X_p, y_p = permute_data(X_train, y_train)
    if method == 'Ridge':
        loss_fn = Ridge_loss
        grad_fn = Ridge_grad
    elif method == 'Lasso':
        loss_fn = Lasso_loss
        grad_fn = Lasso_grad
    else:
        raise ValueError('method = %s is not implemented' % method)

    # train
    for epoch in range(NUM_EPOCHS):
        epoch_loss = AverageMeter()
        for i in range(0, NUM_TRAIN, BATCH_SIZE):
            X_batch = X_p[i: min(i + BATCH_SIZE, NUM_TRAIN)]
            y_batch = y_p[i: min(i + BATCH_SIZE, NUM_TRAIN)]

            loss = loss_fn(X_batch, y_batch, w, LAMBDA)
            epoch_loss.update(loss)
```

```
            if i % 100 == 0 and (not SALIENT):
                print('Epoch [%d/%d] Train: Current Train Loss: %.3f
(Avg Loss %.3f)' % (
                epoch, NUM_EPOCHS, loss, epoch_loss.avg))

            ## -- ! code required
            # perform stochastic gradient descent

        val_loss = validation(X_validation, y_validation, w)
        if not SALIENT:
            print('Epoch [%d/%d] Val: Validation Loss: %.3f' % (epoch,
NUM_EPOCHS, val_loss))
        X_p, y_p = permute_data(X_train, y_train)

    # val
    val_loss = validation(X_validation, y_validation, w)
    if not SALIENT:
        print('Final Validation Loss of %s Regression: %.3f' % (method
, val_loss))
    return val_loss, w


def validation(X, y, w):
    val_loss = mse(X, y, w)
    return val_loss

file_name = "lasso_data_ours.pickle"
with open(file_name, 'rb') as f_myfile:
    data = pickle.load(f_myfile)
    f_myfile.close()

X_train, y_train, X_validation, y_validation = data['x_train'], data['
y_train'], data['x_test'], data['y_test']

## Train and test Ridge Regression

NUM_EPOCHS = 1000
LAMBDA_R = 1
LAMBDA_L = 0.5
BATCH_SIZE = 16
STEP_SIZE = 0.0001
FEAT_SIZE = X_train.shape[1]
INIT_W = np.random.randn(FEAT_SIZE)
print('Training Ridge Regression ...')
loss_ridge,  w_r = training(X_train, y_train, X_validation, y_validati
on, deepcopy(INIT_W), LAMBDA_R, STEP_SIZE, method='Ridge')
print('Training Lasso Regression ...')
loss_lasso,  w_l = training(X_train, y_train, X_validation, y_validati
on, deepcopy(INIT_W), LAMBDA_L, STEP_SIZE, method='Lasso')
print('-' * 50)
print('Final Validation Loss of Ridge Regression: %.3f' % loss_ridge)
print('Final Validation Loss of Lasso Regression: %.3f' % loss_lasso)
```

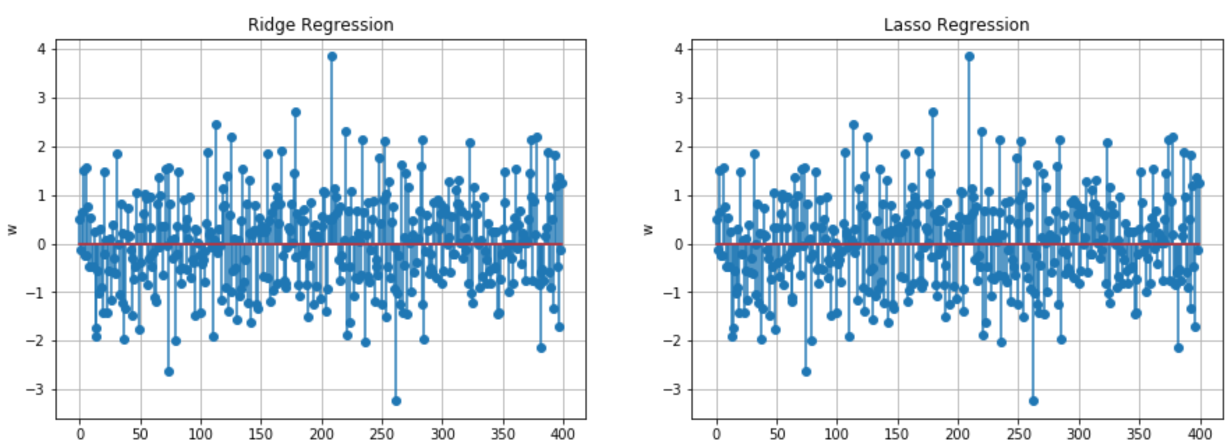
```
# Visualize the learned weight via Ridge Regression and Lasso Regressi
on
fig, ax = plt.subplots(nrows=1, ncols=2)
ax[0].stem(list(range(FEAT_SIZE)), w_r)
ax[0].grid()
ax[0].set_title('Ridge Regression')
ax[0].set_ylabel('w')


ax[1].stem(list(range(FEAT_SIZE)), w_l)
ax[1].grid()
ax[1].set_title('Lasso Regression')
ax[1].set_ylabel('w')

fig.set_size_inches(15, 5)
plt.show()
```

```
Training Ridge Regression ...
Training Lasso Regression ...
--------------------------------------------------
Final Validation Loss of Ridge Regression: 10975.558
Final Validation Loss of Lasso Regression: 10975.558
```

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site
-packages/ipykernel_launcher.py:179: UserWarning: In Matplotlib 3.3
individual lines on a stem plot will be added as a LineCollection in
stead of individual lines. This significantly improves the performan
ce of a stem plot. To remove this warning and switch to the new beha
viour, set the "use_line_collection" keyword argument to True.
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site
-packages/ipykernel_launcher.py:185: UserWarning: In Matplotlib 3.3
individual lines on a stem plot will be added as a LineCollection in
stead of individual lines. This significantly improves the performan
ce of a stem plot. To remove this warning and switch to the new beha
viour, set the "use_line_collection" keyword argument to True.
```



By comparing the learned $\mathbf{w}$ from Ridge Regression and Lasso Regression, we can find Lasso Regression produces sparse $\mathbf{w}$ via L1 regularization.

Q4.3: Hyper-parameter Tuning. Please use different λ_l in the Lasso Regression to find out the best λ_l with the loss on the validation set. Specifically, you should use the validation set to determine the best value for λl and its corresponding validation error/loss.

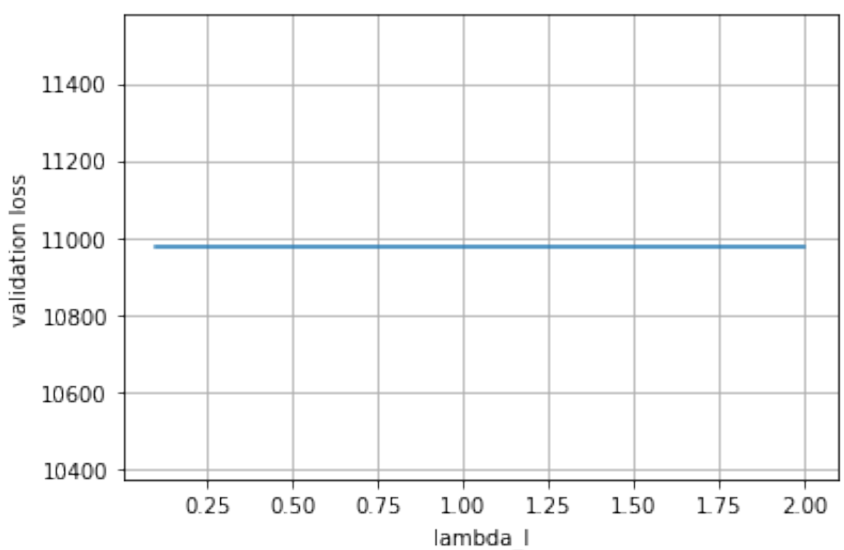
In [4]:

```
# Try different lambdas for Lasso Regression
## -- ! code required
# set different Lambda_L in Lasso Regression
LAMBDA_Ls = [0.1, 0.2, 0.5, 1, 2,]
losses = []
best_lambda_l, best_validation_loss = None, 1e9
for LAMBDA_L in LAMBDA_Ls:
    loss_lasso, w_l = training(X_train, y_train, X_validation, y_valid
ation, deepcopy(INIT_W), LAMBDA_L, STEP_SIZE,
                               method='Lasso')
    if loss_lasso < best_validation_loss:
      best_lambda_l = LAMBDA_L
      best_validation_loss = loss_lasso
    losses.append(loss_lasso)

print(' The best lambda_l is %.3f and its corresponding validation los
s is %.3f' % (best_lambda_l, best_validation_loss))

plt.plot(LAMBDA_Ls, losses)
plt.xlabel('lambda_l')
plt.ylabel('validation loss')
plt.grid()
plt.show()
```

```
 The best lambda_l is 0.100 and its corresponding validation loss is
10975.558
```



In [ ]: