CMPSC 448: Machine Learning. Homework 4b: Calculus and Optimization from Scratch.

Due: February 24, 2020

1. Instructions

- You cannot look at anyone else's code.
- Fill in and upload hw4b.py to gradescope.
- All code (except import statements) in hw4b.py should be inside functions (importing hw4b.py should not cause code to execute).
- Code must have comments and any constants should be stored in a variable defined near the top of your file.

Important! Read this! Your homework **may not** import any python packages. Any attempt to import a package will result in a 0 for the assignment. Do not even use the word "import" or "eval" or "exec" in your file.

2. Some Background

Since your code cannot import any python libraries, you will need a wrapper around it to test your code. For example,

```python
import numpy as np
from matplotlib import pyplot as plt
import pandas as pd
import hw4b

# this code is just a demonstration of functionality
# make sure your code has no free-standing statements

# we will be using the data returned by this function
# the oldfaithful.csv dataset is in the data directory in
# canvas files

def read_data():
    df = pd.read_csv("oldfaithful.csv", sep="\s+", header='infer', index_col=0)
    df.insert(0, 'bias', 1) # add bias as first column
    df.insert(1, 'sq', df.EruptionLength.values**2)
    X =  df[["bias", "sq", "EruptionLength"]].values
    y = df.WaitingTime.values
    return X,y

X, y = read_data()

# how to shuffle data.
# Make sure you have a data matrix X and target y
def makeshuffler(seed=3):
    prng = np.random.RandomState(seed)
    def shuffle(X, y):
        indices = np.arange(y.size)
        prng.shuffle(indices)
        return X[indices], y[indices]
    return shuffle

shuffle = makeshuffler()
X, y = shuffle(X,y) # shuffled dataset

def sigmoid(x):
    result = 1.0/(1 + np.exp(-np.array(x)))
    return result.tolist()
```

```python

def huber(x):
    myval = np.array(x)
    result =  np.where(np.abs(myval) <= 1, 0.5* myval**2, np.abs(myval)-0.5)
    return result.sum()

z = np.arange(-5, 5, 0.1)
plt.plot(z, [huber(a) for a in z])
plt.show() # program will wait until you close the graph

hw4b.question1_loss
```

2.1. **Shuffling the Data.** The code above provides an example of how to shuffle the data. Since you cannot import anything, the shuffle function will be provided as an argument to your code.

2.2. **The sigmoid function.** The sigmoid function, defined above works on numbers, lists, and lists of lists. When x is a single number, this function returns the sigmoid of x. When x is a list or numpy vector, it applies the sigmoid function element-wise (i.e. to each element in the list/vector) and returns a list of the same dimension as x. When x is a list of lists (or numpy matrix), it applies the sigmoid function element-wise and returns a list of lists.

The derivative of the sigmoid is:

$$\frac{d\,\text{sigmoid}(x)}{dx} = \text{sigmoid}(x) * (1 - \text{sigmoid}(x))$$

2.3. **Huber Loss.** The Huber loss, defined above, is an alternative to squared loss and absolute value loss. For numbers close to 0 it acts like the squared loss and for numbers farther away, it acts like absolute value loss. You will need to figure out how to compute its derivative. The formula for huber loss is:

$$\text{huber}(x) = \begin{cases} \frac{1}{2}x^2 & \text{if } |x| \leq 1 \\ |x| - \frac{1}{2} & \text{if } |x| > 1 \end{cases}$$

When applied to a number x, it returns the huber loss of x. When applied to a vector (or list) or a matrix (or list of lists), the result is the sum of the huber losses of the elements.

3. Computing Over Minibatches

Question 1 (5 points)**.** *Suppose our data consists of $(\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n)$. Consider the linear regression model where the prediction is $\widehat{y}_i = \vec{w} \cdot \vec{x}_i$. We can train it with huber loss and regularize, so that the objective function is $\sum_{i=1}^{n} huber(y_i - \widehat{y}_i) + \lambda||\vec{w}||^2$.*

In hw4b.py, fill in the function `question1_loss(w, X, y, indices, lamb, huber)`*. The variable* **w** *will be a 1-dimensional numpy array corresponding to the weight vector.* **X** *will be the data matrix (each row corresponds to a feature vector). The variable* **y** *is the vector of true values. The variable* ***indices*** *is a list of indices of elements in the minibatch. For example, if indices= $[1, 3, 4]$, then the minibatch consists of rows 1, 3, 4 from the matrix X.* **lamb** *is the regularization constant, and* ***huber*** *is a function – the huber loss function.*

Your function should return the minibatch loss. Make sure to properly determine the contribution of each element to the minibatch loss.

Before asking questions, study the slides on optimization and also on linear models. ***Remember, your code may not import anything.***

Question 2 (5 points)**.** *Consider the setting from Question 1.*

In hw4b.py, fill in the function `question2_grad(w, X, y, indices, lamb, huber)`*. The variable* **w** *will be a 1-dimensional numpy array corresponding to the weight vector.* **X** *will be the data matrix (each row corresponds to a feature vector). The variable* **y** *is the vector of true values. The variable* ***indices*** *is a* *list of indices of elements in the minibatch. For example, if indices*$= [1, 3, 4]$*, then the minibatch consists of rows 1, 3, 4 from the matrix* X*.* **lamb** *is the regularization constant, and **huber** is a function – the huber loss function.*

Your function should return the minibatch gradient.

Before asking questions, study the slides on optimization and also on linear models. ***Remember, your code may not import anything.***

Question 3 (5 points)**.** *Consider the setting from Question 1.*

In hw4b.py, fill in the function `question3_update(w, X, y, indices, lamb, eta, huber)`*. The variable* **w** *will be a 1-dimensional numpy array corresponding to the weight vector.* **X** *will be the data matrix (each row corresponds to a feature vector). The variable* **y** *is the vector of true values. The variable **indices** is a list of indices of elements in the minibatch. For example, if indices*$= [1, 3, 4]$*, then the minibatch consists of rows 1, 3, 4 from the matrix* X*.* **lamb** *is the regularization constant, **eta** is the learning rate, and **huber** is a function – the huber loss function.*

Your function should return the updated parameters (i.e. after 1 update of mini-batch gradient descent).

Before asking questions, study the slides on optimization and also on linear models. ***Remember, your code may not import anything.***

Question 4 (10 points)**.** *Consider the setting from Question 1.*

In hw4b.py, fill in the function `question4_n_updates(w, X, y,  lamb, eta, mbatch, n, huber, shuffle)`*. The variable* **w** *will be a 1-dimensional numpy array corresponding to the weight vector.* **X** *will be the data matrix (each row corresponds to a feature vector). The variable* **y** *is the vector of true values.*

*The variable **mbatch** is the minibatch size (the first minibatch is the first mbatch records in the shuffled data, the second minibatch is the next batch records, etc. Note that in the final minibatch you may have fewer than mbatch records).* **lamb** *is the regularization constant, **eta** is the learning rate, **n** is the number of update steps, and **huber** is the huber loss function.*

*The function **shuffle** is the function you should use to shuffle the data.*

Your function should return the updated parameters after n updates.

Before asking questions, study the slides on optimization and also on linear models. ***Remember, your code may not import anything.***

Question 5 (10 points)**.** *Consider the setting from Question 1.*

In hw4b.py, fill in the function `question4_nepochs(w, X, y,  lamb, eta, mbatch, nepochs, huber, shuffle)`*. The variable* **w** *will be a 1-dimensional numpy array corresponding to the weight vector.* **X** *will be the data matrix (each row corresponds to a feature vector). The variable* **y** *is the vector of true values.*

*The variable **mbatch** is the minibatch size (the first minibatch is the first mbatch records in the shuffled data, the second minibatch is the next batch records, etc. Note that in the final minibatch in an epoch you may have fewer than mbatch records).* **lamb** *is the regularization constant, **eta** is the learning rate, **nepochs** is the number of update steps, and **huber** is the huber loss function. The function **shuffle** is the function you should use to shuffle the data.*

*Your function should return the updated parameters after nepochs **epochs**.*

Before asking questions, study the slides on optimization and also on linear models. ***Remember, your code may not import anything.***

4. Training Wheels are Off

Question 6 (35 points)**.** *We are going to be doing linear regression, as in Question 1. However, now we will change the objective function. The ϵ-insensitive squared loss function ℓ_ϵ is defined as:*

$$\ell_\epsilon(z) = \begin{cases} 0 & \text{if } |z| \leq \epsilon \\ (z-\epsilon)^2 & \text{if } z > \epsilon \\ (z+\epsilon)^2 & \text{if } z < -\epsilon \end{cases}$$

The objective function we want to minimize is:

$$\sum_{i=1}^{n} \ell_\epsilon(y_i - \widehat{y}_i) + \lambda ||\vec{w}||^2$$

In hw4b.py, fill in the function `question6_sgd(w, X, y,  lamb, eta, mbatch, nepochs, epsilon, shuffle)`*. The variable* **w** *will be a 1-dimensional numpy array corresponding to the weight vector.* **X** *will be the data matrix (each row corresponds to a feature vector). The variable* **y** *is the vector of true values.*

The variable ***mbatch*** *is the minibatch size (the first minibatch is the first mbatch records in the shuffled data, the second minibatch is the next batch records, etc. Note that in the final minibatch in an epoch you may have fewer than mbatch records).* **lamb** *is the regularization constant,* ***eta*** *is the learning rate,* ***nepochs*** *is the number of update steps.* ***epsilon*** *is the parameter of the ϵ-insensitive loss function. The function* ***shuffle*** *is the function you should use to shuffle the data.*

Your function should return the updated parameters after nepochs ***epochs****.*

Before asking questions, study the slides on optimization and also on linear models. ***Remember, your code may not import anything.***