Introduction to Machine Learning (Fall 2022)

Recitation attendance check

Type in your section passcode to get attendance credit (within the first fifteen minutes of your scheduled section).

Passcode: Enter

100.00%

{0: 'aquarium', 1: 'algae', 2: 'bass', 3: 'coral', 4: 'dartfish', 5: 'eel', 6: 'flatfish', 7: 'guppy', 8: 'hogfish', 9: 'ivy'}

Homework 3 -- Gradient Descent

Due: Wednesday, September 28, 2022 at 11:00 PM

For this homework, it will be helpful to review the following course notes: Regression, and Gradient Descent.

This homework does not have Python code you have to download, but sample test cases can be found in this colab notebook. We encourage you to write your own code to help you answer some of these problems, and/or to test and debug the code components we do ask for.

1) 2D gradient descent

Let's consider gradient descent when the dimension of the input is 2.

$$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$$

So, we have $f(\theta)$ and we are trying to find the values of θ_1 and θ_2 that minimize it. Suppose

$$f(\theta) = -3\theta_1 - \theta_1\theta_2 + 2\theta_2 + \theta_1^2 + \theta_2^2$$

1.1)

What is the first component of $\nabla_\theta f(\theta)$?

Enter an expression involving `theta_1` and `theta_2`. -3 -theta_2 +2*theta_1

Check Syntax | Submit | View Answer **100.00%**

You have infinitely many submissions remaining.

What is the second component of $\nabla_\theta f(\theta)$?

Enter an expression involving `theta_1` and `theta_2`. 2-theta_1 +2*theta_2

Check Syntax | Submit | View Answer **100.00%**

You have infinitely many submissions remaining.

1.2)

What is $f([1, 1])$?

Enter a numerical value. 0

Check Formatting | Submit | View Answer **100.00%**

You have infinitely many submissions remaining.

1.3)

If we started at $\theta = (1, 1)$ and took a step of gradient descent with step-size 0.1, what would the next value of θ be?

Enter a tuple. (1.2, 0.7)

Check Formatting | Submit | View Answer **100.00%**

You have infinitely many submissions remaining.

1.4)

What is $f([1.2, 0.7])$?

Enter a numerical value. -1.11

Check Formatting | Submit | View Answer **100.00%**

You have infinitely many submissions remaining.

1.5)

If we started at $\theta = (1, 1)$ and took a step of gradient descent with step-size 1.0, what would the next value of θ be?

Enter a tuple. (3, -2)

Check Formatting | Submit | View Answer **100.00%**

You have infinitely many submissions remaining.

1.6)

What is $f([3, -2])$?

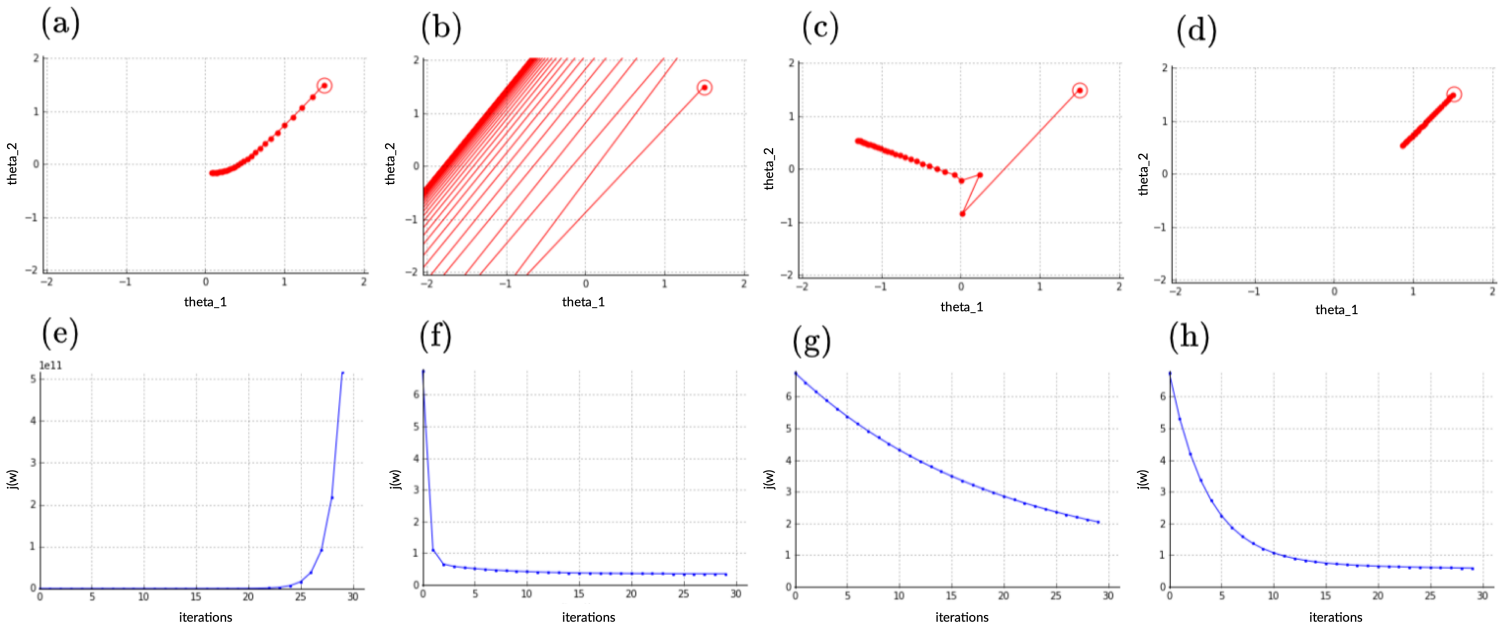
Enter a numerical value. 6

Check Formatting | Submit | View Answer **100.00%**

You have infinitely many submissions remaining.

2) Plotting gradient descent

The following plots show the results of performing linear regression with two parameters using gradient descent with fixed step sizes $0.01, 0.05, 0.50, 1.00$. The plots in the first row illustrate the way the parameters change over time: in each case, the parameters start with values $(1.5, 1.5)$ (illustrated with an extra circle around the point) and we plot points corresponding to the new parameter values after each gradient step (connected with a line to the previous and next parameter values). The plots in the second row show the objective value as a function of iteration number. But, we have mixed up the order of the plots in both rows!



Note that the axes of the plots in the first row are `theta_1` and `theta_2`; the axes of the plots in the second row are the

iteration number (x-axis) and value of the function we are trying to minimize (y-axis).

Which plots in the first row correspond to the weight trajectories for the step sizes 0.01, 0.05, 0.50, 1.00?

Provide a Python list of four strings, e.g. `["a", "b", "c", "d"]`, in the order 0.01, 0.05, 0.50, 1.00.

["d", "a", "c", "b"]

Check Formatting Submit View Answer **100.00%**

You have infinitely many submissions remaining.

Which plots in the second row correspond to the objective for the step sizes 0.01, 0.05, 0.50, 1.00?

Provide a Python list of four strings, e.g. `["e", "f", "g", "h"]`, in the order 0.01, 0.05, 0.50, 1.00.

["g","h","f","e"]

Check Formatting Submit View Answer **100.00%**

You have infinitely many submissions remaining.

3) Alternative loss functions

Although squared loss has a lot of good properties, it can also be very sensitive to "outliers" (points that may have been generated by some kind of underlying error in the data-generation process that we don't really want to model). We'll consider some alternative loss functions.

3.1)

We could just penalize the *absolute difference* between g and a (where g denotes the guess and a denotes the actual label of a data point), so $L_1(g, a) = |g - a|$.

Select all that are true about this loss function:

- [x] It is continuous.
- [ ] Its first derivative w.r.t. g is continuous.
- [x] It penalizes big distances between g and a much less than squared error.

100.00%

You have infinitely many submissions remaining.

Solution:

✓ It is continuous.

✗ Its first derivative w.r.t. g is continuous.

✓ It penalizes big distances between g and a much less than squared error.

Explanation:

1. $|g-a|$ is a composition of continuous functions, so it is continuous.
2. $\frac{d|g-a|}{dg}$ is -1 when $g < a$ and +1 when $g > a$, so there is a discontinuity at $g = a$.*
3. For $u \in \mathbb{R}$ with $|u| > 1$, $|u| < u^2$. For example, $|10 - 110| \ll (10 - 110)^2$.

*Note: $\frac{d|g-a|}{dg}(a)$ is not defined. Although a bit beyond the scope of this class, there are methods to optimize over functions with discontinuous first derivatives. The key takeaway is that the ideas from the rest of this homework still largely apply, but we have to be extra careful about step size to achieve convergence.

3.2)

We could consider an alternative called *pseudo huber loss*, where $L_h(g, a) = \sqrt{1 + (g-a)^2} - 1$.

Select all that are true about this loss function:

- [x] It is continuous.
- [x] Its first derivative w.r.t. g is continuous.
- [x] It penalizes big distances between g and a much less than squared error.

100.00%

You have infinitely many submissions remaining.

Solution:

✓ It is continuous.

✓ Its first derivative w.r.t. g is continuous.

✓ It penalizes big distances between g and a much less than squared error.

Explanation:

1. $\sqrt{1+(g-a)^2}-1$ is a composition of continuous functions, so it is continuous.
2. $\sqrt{1+(g-a)^2}-1$ is a composition of functions with continuous first derivatives, and since both products and compositions of finitely many continuous functions are continuous, applying the chain rule preserves continuity. (Also see next question.)
3. For $|g-a|$ large, $\sqrt{1+(g-a)^2}-1 \approx |g-a| \ll (g-a)^2$.

3.3)

What is $\frac{\partial}{\partial g} L_h(g,a)$?

Enter a Python expression involving `g` and `a`.

Use `**` for exponentiation and `sqrt(x)` for the square root of `x`.

```
1/(sqrt(1+(g-a)**2))*(g-a)
```

Check Syntax | Submit | View Answer **100.00%**

You have infinitely many submissions remaining.

3.4)

If you were doing linear regression with squared loss, without a separate offset and without regularization, the gradient wrt θ would have the form

$$\frac{1}{n}\sum_i f(x^{(i)}, y^{(i)}, \theta)$$

where $f(x, y, \theta) = 2(\theta^T x - y)x$.

If we were using L_h instead, the gradient would have the same form, but with a different function f. Provide a definition for the f associated with L_h.

Enter a Python expression involving `x`, `y`, and `theta`.

Use $**$ for exponentiation, `sqrt(x)` for the square root of `x`, `transpose(x)` for the transpose of `x`, and `p@q` for the matrix product of `p` and `q`.

```
x@(transpose(theta)@x - y)/sqrt(1+(transpose(theta)@x - y)**2)
```

100.00%

You have infinitely many submissions remaining.

Solution: `((transpose(theta) @ x) - y) * x / sqrt(1 + (transpose(theta)@x - y)**2)`

$$\frac{\left((\theta)^T \mathbf{x} - \mathbf{y}\right) \times \mathbf{x}}{\sqrt{1 + \left((\theta)^T \mathbf{x} - \mathbf{y}\right)^2}}$$

Explanation:

$$L_h(\theta^T x, y) = \left(1 + (\theta^T x - y)^2\right)^{1/2} - 1 \tag{1}$$

$$\frac{\partial}{\partial \theta} L_h(\theta^T x, y) = (1/2)\left(1 + (\theta^T x - y)^2\right)^{-1/2} (2(\theta^T x - y)x) \tag{2}$$

$$= \frac{\left(\theta^T x - y\right) x}{\sqrt{1 + (\theta^T x - y)^2}}. \tag{3}$$

4) Hypothesis classes

For each of the following hypothesis classes, write down an expression for the partial derivative of the *squared loss* L_2 on a single example (x, y) with respect to parameter vector θ using the symbols: `x`, `y` and `theta`. Remember that you can use $*$ for element-wise multiplication, `@` for dot product, and `transpose(a)` for transpose a vector or array:

4.1)

$$h(x;\theta) = \theta^T x + \theta^T x^2$$

Note that x^2 here is computed from the input vector by squaring each component (i.e. $x^2 = x * x$).

Enter an expression for $\nabla_\theta L_2(x, y)$. `2*(transpose(theta)@x + tr`

Check Syntax | Submit | View Answer **100.00%**

You have infinitely many submissions remaining.

4.2)

$$h(x;\theta) = (\theta^T x)^2$$

Enter an expression for $\nabla_\theta L_2(x, y)$. `4*(transpose(theta)@x)**3`

Check Syntax | Submit | View Answer **100.00%**

You have infinitely many submissions remaining.

4.3)

$$h(x;\theta) = (\theta^2)^T x$$

Note that θ^2 here is computed from the input vector by squaring each component (i.e. $\theta^2 = \theta * \theta$).

Enter an expression for $\nabla_\theta L_2(x, y)$. `4*(transpose(theta*theta)@`

Check Syntax | Submit | View Answer **100.00%**

You have infinitely many submissions remaining.

4.4)

Indicate which, if any, of these hypothesis classes can represent all of the hypotheses that can be represented by usual linear regression:

Choose all that's correct:

- [ ] $h(x;\theta) = \theta^T x + \theta^T x^2$
- [ ] $h(x;\theta) = (\theta^T x)^2$
- [x] $h(x;\theta) = (\theta^3)^T x$

Submit View Answer 100.00%

You have infinitely many submissions remaining.

5) Stochastic gradient descent for ridge regression

Recall that the ridge-regression objective is

$$J_{\text{ridge}}(\theta, \theta_0) = \left(\frac{1}{n} \sum_{i=1}^{n} \left(\theta^T x^{(i)} + \theta_0 - y^{(i)} \right)^2 \right) + \lambda \|\theta\|^2$$

We would like to do stochastic gradient descent to optimize this objective on a data set. To apply SGD, we need to describe our objective in the form

$$\sum_i f(x^{(i)}, y^{(i)}; \theta, \theta_0)$$

Provide a Python expression for $f(x, y; \theta, \theta_0)$ that makes this objective equivalent to J_{ridge}. Your expression should include $\theta, \theta_0, x, y, \lambda, n$. You will also need to use transpose, @ and * appropriately.

```
(transpose(theta)@x + theta_0 - y)**2/n + lambda*transpose(theta)@theta/n
```

Check Syntax Submit View Answer 100.00%

You have infinitely many submissions remaining.

6) Stochastic gradient descent in 1D

Consider a 1D regression problem without an offset θ_0, which forces the hypothesis to go through the point $x = 0, y = 0$! There is one parameter θ in this model, and our hypothesis is

$$h(x;\theta) = \theta x$$

where we can treat x and θ as scalar for simplicity.

Our data set has two points:

$$\mathcal{D} = \{(1, 1), (1, -1)\}$$

6.1)

What value of θ minimizes squared loss on this data set?

Enter a numeric value. 0

Check Formatting Submit View Answer **100.00%**

You have infinitely many submissions remaining.

6.2)

Starting with an initial guess of $\theta = 1$, and letting step size be $\eta = 0.2$, what are the values of θ after the first step of batch gradient descent and after the second steps of batch gradient descent?

Enter your answer as a list of θ values of the form `[a,b]`. [0.6, 0.36]

Check Formatting Submit View Answer **100.00%**

You have infinitely many submissions remaining.

6.3)

Now let's consider a variation on SGD, but instead of randomly picking a data point to train on, we will strictly alternate between our two data points.

We'll start with $\theta = -1/9$ and use a fixed-sized $\eta = 0.2$. Assume that you start with the data point $(1, 1)$. Now, what are the first four steps?

Assume that SGD takes a step of the form:

$$\theta = \theta - \eta \frac{2}{n}(\theta x^i - y^i)x^i$$

Enter your answer as a list of θ values. Use a fraction if possible.

[1/9, -1/9, 1/9, -1/9]

Check Formatting Submit View Answer **100.00%**

You have infinitely many submissions remaining.

6.4)

Hmm. Maybe our problem is that we need a smaller step size. Ally argues that no matter what fixed step size we pick, there

is an initial θ that will simply oscillate back and forth and not converge to the optimal θ.

Prove Ally correct by writing an expression for θ, in terms of η, that has the property that if we start with initial parameter value θ then after an update on the first training point, we will end up with parameter value $-\theta$. Assume that the first data point is $(1, 1)$.

Again, assume we will take an SGD step of the form:

$$\theta = \theta - \eta \frac{2}{n}(\theta x^i - y^i)x^i$$

Enter an expression for θ in terms of η. Use the string `eta` when refering to η. eta/(eta-2)

Check Syntax Submit View Answer 100.00%

You have infinitely many submissions remaining.

6.5)

How can we fix the problem raised by Ally? **Hint:** Think about what the infinite sum of $\eta(t)$ would be.

Choose one:

- Choose points at random for updates rather than alternating between them
- Instead of constant step size, let it depend on time as $\eta(t) = t$
- Instead of constant step size, let it depend on time as $\eta(t) = .99^t$
- Instead of constant step size, let it depend on time as $\eta(t) = 1/t$
- Don't choose examples with troublesome symmetries

100.00%

You have infinitely many submissions remaining.

Solution: Instead of constant step size, let it depend on time as $\eta(t) = 1/t$

Explanation:

Repeatedly trying out random initialization points until we find one that doesn't lead to an oscillating descent might work but feels inefficient. If possible we would like to come up with a learning rate schedule $\eta(t)$ such that no matter where we start, the algorithm probably converges as $t \to \infty$. At each update we take a step of size $\delta(t) = \eta(t)\|\nabla_\theta J(\theta_t)\|$, so to achieve convergence we must have

$$\lim_{t \to \infty} \delta(t) = 0.$$

At any particular t we don't have much control over $\|\nabla_\theta J(\theta_t)\|$, but we have full control over $\eta(t)$, so it makes sense to choose $\eta(t)$ such that $\eta(t) \to 0$ as $t \to \infty$. In particular the choice $\eta(t) = t$ is out, leaving 0.99^t and $1/t$ under consideration.

How fast should $\eta(t)$ decrease? Suppose $\eta(t) = 0.99^t$ and we take T steps. Furthermore, suppose $\|\nabla_\theta J(\theta_t)\|$ is bounded above by some finite positive constant K (aka K-Lipschitz). Then we have that

$$\begin{aligned}
\sum_{t=0}^{T} \delta(t) &= \sum_{t=1}^{T} \eta(t) \|\nabla J(\theta_t)\| && (4)\\
&\leq \sum_{t=0}^{T} \eta(t) K && (5)\\
&= K \sum_{t=0}^{T} \eta(t) && (6)\\
&= K \sum_{t=0}^{T} 0.99^t && (7)\\
&< K \sum_{t=0}^{\infty} 0.99^t && (8)\\
&= K \frac{1}{1 - 0.99} && (9)\\
&= 100K, && (10)
\end{aligned}$$

so even if T were the age of the universe we could never leave the sphere of radius $100K$ centered at θ_0. We can't just expect to get lucky and always start with θ_0 less than $100K$ distance away from the optimum θ^*, so we would like to have

$$\sum_{t=0}^{\infty} \eta(t) = \infty, \quad (11)$$

which is satisfied by the choice $\eta(t) = 1/t$.

7) Fair Regression

Fairley is interested in using linear regression to make a prediction, but they have data from two different populations, $\mathcal{D}_1$ and $\mathcal{D}_2$, and they would prefer that the hypothesis have similar accuracy in both populations. There are $n_{\mathcal{D}_1}$ points in the first data set and $n_{\mathcal{D}_2}$ in the second. The x values in both data sets have dimension $d \times 1$.

7.1)

Fairley decides to use the following training objective(cost), with the idea that the last term would account for fairness:

$$J_F(\theta, \theta_0) = \mathcal{L}(\theta, \theta_0; \mathcal{D}_1) + \mathcal{L}(\theta, \theta_0; \mathcal{D}_2) + \lambda(\mathcal{L}(\theta, \theta_0; \mathcal{D}_1) - \mathcal{L}(\theta, \theta_0; \mathcal{D}_2))^2$$

What sign should λ have, in order to pressure the value of $\mathcal{L}(\theta, \theta_0; \mathcal{D}_1)$ and $\mathcal{L}(\theta, \theta_0; \mathcal{D}_2)$ to be similar?

- Negative
- Positive

100.00%

You have infinitely many submissions remaining.

Solution: Positive

Explanation:

A positve value of λ will add a positive contribution to the overall loss. This contribution depends on the squared differences between $\mathcal{L}(\theta, \theta_0; \mathcal{D}_1)$ and $\mathcal{L}(\theta, \theta_0; \mathcal{D}_2)$. We are solving a minimization problem and therefore the sum of the squared differences will be minimized. The values of $\mathcal{L}(\theta, \theta_0; \mathcal{D}_1)$ and $\mathcal{L}(\theta, \theta_0; \mathcal{D}_2)$ will be similar if their squared difference is small.

7.2)

If we set λ to have a **very** large magnitude, and used gradient descent to find the θ, θ_0 that optimize J_F, which of these would be true of the resulting hypothesis?

- It would **definitely** have **small** values of $\mathcal{L}(\theta, \theta_0; \mathcal{D}_1)$ and $\mathcal{L}(\theta, \theta_0; \mathcal{D}_2)$
- It **might** have **large** values of $\mathcal{L}(\theta, \theta_0; \mathcal{D}_1)$ and $\mathcal{L}(\theta, \theta_0; \mathcal{D}_2)$
- It is **likely** to have at least one **small** value of $\mathcal{L}(\theta, \theta_0; \mathcal{D}_1)$ or $\mathcal{L}(\theta, \theta_0; \mathcal{D}_2)$.

100.00%

You have infinitely many submissions remaining.

Solution: It **might** have **large** values of $\mathcal{L}(\theta, \theta_0; \mathcal{D}_1)$ and $\mathcal{L}(\theta, \theta_0; \mathcal{D}_2)$

Explanation:

If we set λ to a very large magnitude then the values of $\mathcal{L}(\theta, \theta_0; \mathcal{D}_1)$ and $\mathcal{L}(\theta, \theta_0; \mathcal{D}_2)$ will not matter anymore. Only their squared difference will be important. Therefore, these values can be potentially large.

7.3)

If we set λ to have a **very** large magnitude, and used gradient descent to find the θ, θ_0 that optimize J_F, which of these would be true of the resulting hypothesis?

- (●) It would **definitely** have close to the **same** values of $\mathcal{L}(\theta, \theta_0; \mathcal{D}_1)$ and $\mathcal{L}(\theta, \theta_0; \mathcal{D}_2)$
- ( ) It **might** have close to the **same** values of $\mathcal{L}(\theta, \theta_0; \mathcal{D}_1)$ and $\mathcal{L}(\theta, \theta_0; \mathcal{D}_2)$
- ( ) It would **definitely** have substantially **different** values of $\mathcal{L}(\theta, \theta_0; \mathcal{D}_1)$ and $\mathcal{L}(\theta, \theta_0; \mathcal{D}_2)$.

100.00%

You have infinitely many submissions remaining.

Solution: It would **definitely** have close to the **same** values of $\mathcal{L}(\theta, \theta_0; \mathcal{D}_1)$ and $\mathcal{L}(\theta, \theta_0; \mathcal{D}_2)$

Explanation:

If we set λ to have a **very** large magnitude then the optimization will force the $\mathcal{L}(\theta, \theta_0; \mathcal{D}_1)$ and $\mathcal{L}(\theta, \theta_0; \mathcal{D}_2)$ to have a small value of their squared differences. Therefore, they will be similar.

7.4)

Assume we have test data set $\mathcal{T}_1$ drawn from the same population as $\mathcal{D}_1$ and test data set $\mathcal{T}_2$ drawn from the same population as $\mathcal{D}_2$. Would you expect that putting a greater emphasis on the fairness term in the training objective J_F would:

- ( ) **increase** prediction accuracy of the resulting hypothesis on **both** $\mathcal{T}_1$ and $\mathcal{T}_2$
- (●) **decrease** prediction accuracy of the resulting hypothesis on **at least one of** $\mathcal{T}_1$ and $\mathcal{T}_2$.
- ( ) Neither (i.e. have none of the effects listed above).

Submit View Answer **100.00%**

You have infinitely many submissions remaining.

7.5)

Assume we have test data set $\mathcal{T}_1$ drawn from the same population as $\mathcal{D}_1$ and test data set $\mathcal{T}_2$ drawn from the same population as $\mathcal{D}_2$. Would you expect that putting a greater emphasis on the fairness term in the training objective J_F would:

- (●) make prediction accuracy of the resulting hypothesis on $\mathcal{T}_1$ **more similar** to its accuracy on $\mathcal{T}_2$
- ( ) make prediction accuracy of the resulting hypothesis on $\mathcal{T}_1$ **more different** from its accuracy on $\mathcal{T}_2$
- ( ) have **no effect** on the relative prediction accuracy of the resulting hypothesis on the two testing sets

100.00%

You have infinitely many submissions remaining.

Solution: make prediction accuracy of the resulting hypothesis on $\mathcal{T}_1$ **more similar** to its accuracy on $\mathcal{T}_2$

Explanation:

Increasing the value of λ will make $\mathcal{L}(\theta, \theta_0; \mathcal{D}_1)$ and $\mathcal{L}(\theta, \theta_0; \mathcal{D}_2)$ similar when evaluated on the training dataset. We expect this to be true also on the testing datasets.

7.6)

What is $\nabla_\theta J_F(\theta, \theta_0)$?

Write an expression for $\nabla_\theta J_F(\theta, \theta_0)$, using:

L1 to stand for $\mathcal{L}(\theta, \theta_0; \mathcal{D}_1)$,
L2 to stand for $\mathcal{L}(\theta, \theta_0; \mathcal{D}_2)$,
G1 to stand for $\nabla_\theta \mathcal{L}(\theta, \theta_0; \mathcal{D}_1)$,
G2 to stand for $\nabla_\theta \mathcal{L}(\theta, \theta_0; \mathcal{D}_2)$,
lambda to stand for λ
$*$ for multiplication
+ for addition
− for subtraction

G1 + G2 +2*lambda*(G1-G2)*(L1-L2)

Check Syntax | Submit | View Answer **100.00%**

You have infinitely many submissions remaining.

7.7)

What is the dimension of $\nabla_\theta J_F(\theta, \theta_0)$?

- $n \times 1$
- $d \times n$
- $d \times 1$
- $d \times d$
- $d \times 2d$
- $2d \times 1$

Submit View Answer 100.00%

You have infinitely many submissions remaining.

8) Implementing gradient descent

In this section we will implement generic versions of gradient descent and apply these to the linear regression (with regularization) objective.

Reminder: For your convenience, you may use the colab notebook (linked on the top of the page) to debug your code.

8.1) Gradient descent

Note: If you need a refresher on gradient descent, you may want to reference Gradient Descent.

We want to find the x that minimizes the value of the *objective function* $f(x)$, for an arbitrary scalar function f. The function f will be implemented as a Python function of one argument, that will be a numpy column vector. For efficiency, we will work with Python functions that return not just the value of f at x, $f(x)$, but also return the gradient vector at x, that is, $\nabla_x f(x)$.

We will now implement a generic gradient descent function, `gd`, that has the following input arguments:

- `f`: a function whose input is a `x`, a column vector, and returns a scalar.
- `df`: a function whose input is a `x`, a column vector, and returns a column vector representing the gradient of `f` at `x`.
- `x0`: an initial value of x, which is a column vector.
- `step_size_fn`: a function that is given the iteration index (an integer starting at zero) and returns a step size.
- `num_steps`: the number of update steps to perform

Our function `gd` returns a tuple:

- `x`: the value at the final step
- `fx`: the value of `f(x)` at the final step

Hint: This is a short function!

Some test or example functions (made available to you via the code and colab referenced at the top of this homework page) that you may find useful are reproduced below.

You may also find `rv` and `cv` (from previous weeks) useful, though not necessary.

```
def f1(x): # f(x,y,z) = y^2 + z
    return x[1:2, :]**2 + x[2:3, :]

def df1(x):
    x = list(x.squeeze())
    return cv([0, 2*x[1], 1])

def f2(x): # f(x,y,z) = xy
    return x[0:1, :]*x[1:2, :]

def df2(x):
    x = list(x.squeeze())
    return cv([x[1], x[0], 1])

def rv(values):
    return np.array([values])

def cv(values):
    return rv(values).T
```

To evaluate results, we also use a simple `package_ans` function, which checks the final `x` and `fx` values.

```
def package_ans(gd_vals):
    x, fx = gd_vals
    return [x.tolist(), fx.tolist()]
```

The test cases are provided below, but you should feel free (and are encouraged!) to write more of your own.

```
# Test case 1
ans=package_ans(gd(f1, df1, cv([1.,1.,1.]), lambda i: 0.1, 1000))

# Test case 2
ans=package_ans(gd(f2, df2, cv([2., 3., 4.]), lambda i: 0.01, 1000))
```

```
def gd(f, df, x0, step_size_fn, num_steps):
    i = 0
    while i < num_steps:
        x0 = x0 - step_size_fn(i)*df(x0)
        i+= 1
    return x0, f(x0)
```

Run Code | Submit | View Answer **100.00%**

You have infinitely many submissions remaining.

8.2) Numerical Gradient

Getting the analytic gradient correct for complicated functions is tricky. A very handy method of verifying the analytic gradient or even substituting for it is to estimate the gradient at a point by means of *finite differences*.

Assume that we are given a function $f(x)$ that takes a column vector as its argument and returns a scalar value. In gradient descent, we will want to estimate the gradient of f at a particular x_0.

The i^{th} component of $\nabla_x f(x_0)$ can be estimated as

$$\frac{f(x_0 + \delta^i) - f(x_0 - \delta^i)}{2\delta}$$

where δ^i is a column vector whose i^{th} coordinate is δ, a small constant such as 0.001, and whose other components are zero. Note that adding or subtracting δ^i is the same as incrementing or decrementing the i^{th} component of x_0 by δ, leaving the other components of x_0 unchanged. Using these results, we can estimate the i^{th} component of the gradient.

For example, if $x_0 = (1, 1, \ldots, 1)^T$ and $\delta = 0.01$, we may approximate the first component of $\nabla_x f(x_0)$ as

$$\frac{f((1,1,1,\ldots)^T + (0.01,0,0,\ldots)^T) - f((1,1,1,\ldots)^T - (0.01,0,0,\ldots)^T)}{2 \cdot 0.01}.$$

(We add the transpose so that these are column vectors.) **This process should be done for each dimension independently, and together the results of each computation are compiled to give the estimated gradient, which is d dimensional.**

Implement this as a function `make_num_grad_fn` that takes as arguments the objective function `f` and a value of `delta`, and returns a new **function** that takes an `x` (a column vector of parameters) and returns a gradient column vector.

Depending on your implementation strategy, an observation that may be useful to you is that although the input `x` is intended to be a $d \times 1$ column vector, the test-case functions `f1, f2` are implemented such that they are also compatible with $d \times n$ matrix input, on which they compute the same function on each column of the matrix, returning a $1 \times n$ row vector of function outputs.

Hint 1: All of your code will be within the function `df`. Can you see why? See this for more information on returning a function instead of a value.

Hint 2: If you do `temp_x = x` where `x` is a vector (numpy array), then `temp_x` is just another name for the same vector as `x` and changing an entry in one will change an entry in the other. You should either use `x.copy()` or remember to change entries back after modification.

Optional Challenge: Can you implement this without using loops?

The test cases are shown below; these use the functions defined in the previous exercise. **Tip:** work out a simple 1D example and test your code (with print statements) to help with debugging!

```
x = cv([1., 1., 1.])
ans=(make_num_grad_fn(f1)(x).tolist(), x.tolist())

x = cv([1.,2.,3.])
ans=(make_num_grad_fn(f1)(x).tolist(), x.tolist())

x = cv([-1., -1., -1.])
ans=(make_num_grad_fn(f2)(x).tolist(), x.tolist())

x = cv([-1., -2., -3.])
ans=(make_num_grad_fn(f2)(x).tolist(), x.tolist())
```

```python
def make_num_grad_fn(f, delta=0.001):
    def df(x):
        vector = [0]*len(x)
        for i in range(len(x)):
            temp = [0]*len(x)
            temp[i] = delta
            temp = np.array([temp]).T
            vector[i] = float((f(x+temp) - f(x-temp))/(2*delta))
        return np.array([vector]).T
    return df

```

Run Code Submit View Answer 100.00%

You have infinitely many submissions remaining.

A faster (one function evaluation per entry), though sometimes less accurate, estimate is to use:

$$\frac{f(x_0 + \delta^i) - f(x_0)}{\delta}$$

for the i^{th} component of $\nabla_x f(x_0)$.

8.3) Using the Numerical Gradient

Recall that our generic gradient descent function takes both a function `f` that returns the value of our function at a given point, and `df`, a function that returns a gradient at a given point. Write a function `minimize` that takes only a function `f` and uses this function and numerical gradient descent to return the local minimum (both `x` and `f(x)`).

In the Colab, your implementation of `minimize` should call your own implementations of `gd` and `make_num_grad_fn` (that you will need to copy into the corresponding cells). However, when checking for correctness below, we use our own implementations of `gd` and `make_num_grad_fn` which are the same as our provided solutions to the parts above (only "View Answer" once you have submitted your answer and got credit for your solutions).

You do not need to provide implementations for `gd` and `make_num_grad_fn` below. You may use the default of `delta=0.001` for `make_num_grad_fn`.

Hint: Your definition of `minimize` should call `make_num_grad_fn` exactly once to return a function. Then you may call this function many times in your updates for numerical gradient descent. You should return the same outputs as `gd`.

The test cases are:

```
ans = package_ans(minimize(f1, cv([1.,1.,1.]), lambda i: 0.1, 1000))
```

```
ans = package_ans(minimize(f2, cv([2., 3., 4.]), lambda i: 0.01, 1000))
```

```python
def minimize(f, x0, step_size_fn, num_steps):
    """
    Parameters:
      See definitions in 4.1
    Returns:
      same output as gd, i.e. (x, f(x))
    """
    func = make_num_grad_fn(f, delta = 0.001)
    return gd(f, func, x0, step_size_fn, num_steps)


```

Run Code Submit View Answer 100.00%

You have infinitely many submissions remaining.

9) Stochastic gradient

We will now write some general python code to implement **stochastic gradient descent**.

`sgd` takes the following as input: (Recall that the *stochastic* part refers to using a randomly selected point and corresponding label from the given dataset to perform an update. Therefore, your objective function for a given step will need to take this into account.)

- `X`: a standard data array (d by n)
- `y`: a standard labels row vector (1 by n)
- `J`: a cost function whose input is a data point (a column vector), a label (1 by 1) and a weight vector `w` (a column vector) (in that order), and which returns a scalar.
- `dJ`: a cost function gradient (corresponding to `J`) whose input is a data point (a column vector), a label (1 by 1) and a weight vector `w` (a column vector) (also in that order), and which returns a column vector.
- `w0`: an initial value of weight vector w, which is a column vector.
- `step_size_fn`: a function that is given the (zero-indexed) iteration index (an integer) and returns a step size.
- `max_iter`: the number of iterations to perform

It returns the following:

- `w`: the value of the weight vector at the final step (same dimensions as the input `w0`).

  You might find the function `np.random.randint(n)` useful in your implementation.

  Hint: This is a short function; our implementation is around 10 lines.

  Along with testing on Colab, you can submit your answer on the answer checker here for debugging because it will provide helpful hints for correcting common mistakes.

```
def sgd(X, y, J, dJ, w0, step_size_fn, max_iter):
    n = len(y[0])
    print(len(X))
    for i in range(max_iter):
        temp = np.random.randint(n)
        w0 = w0 - step_size_fn(i)*dJ(X[:, temp:temp+1], y[:, temp:temp+1], w
    return w0
```

Run Code | Submit | View Answer **100.00%**

You have infinitely many submissions remaining.

10) Tying Everything Together

Last week, we defined the _ridge regression_ objective function.

$$J_{\text{ridge}}(\theta, \theta_0) = \frac{1}{n}\sum_{i=1}^{n}\left(\theta^T x^{(i)} + \theta_0 - y^{(i)}\right)^2 + \lambda\|\theta\|^2$$

Recall that in last week's homework, we derived the closed-form solution for the θ value that minimizes the *least squares* loss function. However, it is computationally challenging to compute the closed-form solution on θ on high-dimensional data and large datasets.

This week, we wish to apply gradient descent and stochastic gradient descent to minimize the ridge regression function. In order to use the gradient descent and stochastic gradient descent functions implemented previously in the homework, we must add ones to the end of each datapoint in X, which we implemented in the `add_ones_row` function in homework 1.

10.1) Gradient Descent and Stochastic Gradient Descent for Linear Regression

In the next subsections, we assume that X is a $d \times n$ matrix, Y is a $1 \times n$ matrix, and θ is a $d \times 1$ matrix. Rewriting the ridge objective through matrix operations, we find that:

$$J_{\text{ridge}}(\theta) = \frac{1}{n}(\theta^T X - Y)(\theta^T X - Y)^T + \lambda||\theta||^2$$

For the rest of the problem, assume that `X` already has ones at the end of each datapoint. You **do not** need to call the `add_one_rows` function.

10.1.1)

Write an expression for $\nabla J_{\text{ridge}}(\theta)$ with respect to θ.

Enter your answers as mathematical expressions. You should use `transpose(m)` for transpose of an array `m`, `f(q)` for a function f applied to scalar or vector `x`, and `p@q` to indicate a matrix product of two arrays/matrices, `p` and `q`. Remember that `p*q` denotes component-wise multiplication.

Enter a Python expression involving `X`, `Y`, `lambda`, `n`, and `theta`. You will also need to use `transpose`, `@` and $*$ appropriately. 2/n*X@transpose((transpose(theta)@X - Y)) + 2*lambda*theta

Check Syntax Submit View Answer 100.00%

You have infinitely many submissions remaining.

10.1.2)

Implement `objective_func`. `objective_func` returns a function that computes $J_{\text{ridge}}(\theta)$.

```
inputs:
    X: a (dxn) numpy array.
    Y: a (1xn) numpy array
    lam: regularization parameter
outputs:
    f : a function that takes in a (dx1) numpy array "theta" and returns *as a float* the
value of the ridge regression objective when theta(the variable)="theta"(the numpy array)
```

```python
def objective_func(X, Y, lam):
    def f(theta):
        # write your implementation here
        print(theta)
        n = np.shape(Y)[1]
        print(1/n*((theta.T@X - Y)@(theta.T@X - Y).T) + lam*theta.T@theta)
        return float(1/n*((theta.T@X - Y)@(theta.T@X - Y).T) + lam*theta.T@t
    return f


```

Run Code Submit View Answer 100.00%

You have infinitely many submissions remaining.

10.1.3)

Implement `objective_func_grad`. `objective_func_grad` returns a function that computes $\nabla J_{\text{ridge}}(\theta)$.

```
inputs:
    X: a (dxn) numpy array.
    Y: a (1xn) numpy array
    lam: regularization parameter
outputs:
    df : a function that takes in a (dx1) numpy array "theta" and returns the gradient of
the ridge regression objective when theta(the variable)="theta"(the numpy array)
```

```python
def objective_func_grad(X, Y, lam):
    def df(theta):
        n = np.shape(Y)[1]
        print(2/n*(X@(theta.T@X - Y).T) + 2*lam*theta)
        return 2/n*(X@(theta.T@X - Y).T) + 2*lam*theta
    return df

```

Run Code | Submit | View Answer **100.00%**

You have infinitely many submissions remaining.

10.2) Finding the best parameters (Optional: all subparts below)

So far in the course, you've learned about two different hyperparameters: the regularization rate λ and the step size/learning rate η. You might be wondering, how do we pick the best hyperparameters for minimizing the loss function?

One of the most basic ways to pick regularization rate and learning rate is to use grid search. The basic idea behind grid search is to select several possible (λ, η) pairs, train models with every combination of these hyperparameters, and evaluate each trained model to select the best hyperparameters.

10.2.1) Check your understanding

What is the best method for evaluating the hyperparameters (λ, η) from these available options? Assume that all of these options are computationally feasible.

- (●) Use k-fold cross-validation
- ( ) Divide data into a training, testing, and validation set. Train the model once on the training dataset and use the loss on the **training set**.
- ( ) Use $\lambda + \eta$

Submit View Answer **100.00%**

You have infinitely many submissions remaining.

What values of λ, η are used during *evaluation*, i.e. when computing the error?

- (●) We do not use λ and η during evaluation
- ( ) 0.01, 0.01
- ( ) 1, 5
- ( ) The best values found through grid search.

Submit View Answer **100.00%**

You have infinitely many submissions remaining.

10.2.2) Running Grid Search

We will be running grid search over the Boston Housing dataset. For more information about this dataset, please visit this link.

For the rest of this exercise, we will be predicting the median value of houses in the Boston area using linear regression and gradient descent. Please visit the colab notebook linked on the top of this page to collect metrics on how well gradient descent works on this regression problem.

Among the grid of values specified in the colab, what is the best value of λ and η when using gradient descent? Enter your answer as a tuple (λ, η)

Check Formatting Submit View Answer

You have infinitely many submissions remaining.

What is the *test set* error using the best (λ, η) combination for the **gradient descent** case? Enter your answer up to 3 decimal places.

Check Formatting Submit View Answer

You have infinitely many submissions remaining.

What is the *test set* error using the best λ for the **analytic** case? Enter your answer up to 3 decimal places.

Check Formatting Submit View Answer

You have infinitely many submissions remaining.