Chapter 3- Optimization Exercises

Exercise 1

We have $f(x) = x^2$, and we take the learning rate $\alpha = 0.2$ and the starting point $x_0 = 3$.

Compute the first three iterations of gradient descent, where does it converge to?

Solution

- Gradient of $f(x) = x^2$:

$$\nabla f(x) = 2x$$

Remember that the **Gradient Descent update rule** is $x_{n+1} = x_n - \alpha \nabla f(x_n)$.

- Now we can start with the iterations (starting from $x_0 = 3$):

$$\nabla f(3) = 2(3) = 6$$
$$x_1 = 3 - 0.2 \cdot 6 = 1.8$$

Continuing with the next iterations:

Iteration	x_n	$\nabla f(x_n)$	Update Calculation	x_{n+1}
0	3.00	6.00	$3 - 0.2 \cdot 6$	1.80
1	1.80	3.60	$1.8 - 0.2 \cdot 3.6$	1.08
2	1.08	2.16	$1.08 - 0.2 \cdot 2.16$	0.864

- **Convergence**:

The function $f(x) = x^2$ is **convex**, and the minimum is achieved at:

$$x^* = 0$$

The iterations are clearly **converging toward** 0 .

Python code

```python
def f(x): # objective function
    return x**2

def grad_f(x): # gradient of the objective function
    return 2 * x

alpha = 0.2 # learning rate
x0 = 3 # starting point
iterations = 3 # number of iterations

x_values = [x0] # list to store x values

for n in range(iterations):
    x_n = x_values[-1] # last element in the list
    grad = grad_f(x_n) # compute gradient
    x_next = x_n - alpha * grad # update step
    print(
        f"Iteration {n}: "
        f"x_n = {x_n:.4f}, "
        f"f'(x_n) = {grad:.4f}, "
        f"x_(n+1) = {x_next:.4f}"
    )
    x_values.append(x_next) # append new value to the list
```

Exercise 2

We have

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{n}(\theta - a_i)^2, \quad a = [-1, 1, 0, 2]$$

We use **Full-Batch Gradient Descent** with **learning rate** $\alpha = 0.1$ and **starting point** $\theta_0 = 3$

- compute the first **three iterations**
- Where does it converge to?

Solution

- First, compute the **mean** of the data:

$$\bar{a} = \frac{-1+1+0+2}{4} = 0.5$$

- Gradient of the loss ($L(\theta) = \frac{1}{n}\sum_{i=1}^{n}(\theta - a_i)^2$), Taking the derivative w.r.t θ :

$$\nabla L(\theta) = \frac{2}{n}\sum_{i=1}^{n}(\theta - a_i) = 2(\theta - \bar{a})$$

With $\bar{a} = 0.5$:

$$\nabla L(\theta) = 2\theta - 1$$

Remember that the **Gradient Descent update rule** is $\theta_{n+1} = \theta_n - \alpha \nabla L(\theta_n)$.

- Now we can start with the iterations (starting from $\theta_0 = 3$):

$$\nabla L(3) = 2(3) - 1 = 5$$
$$\theta_1 = 3 - 0.1 \cdot 5 = 2.5$$

Continuing with the next iterations:

Iteration	θ_n	$\nabla L(\theta_n)$	Update Calculation	θ_{n+1}
0	3.00	5.00	$3 - 0.1 \cdot 5$	2.50
1	2.50	4.00	$2.5 - 0.1 \cdot 4$	2.10
2	2.10	3.20	$2.1 - 0.1 \cdot 3.2$	1.78

- **Convergence**:

The loss function is **convex**, and the minimum is achieved at:

$$\theta^* = \bar{a} = 0.5$$

The iterations are clearly **converging toward** 0.5 .

Python code

```python
def L(theta, a):
    return sum((theta - ai)**2 for ai in a) / len(a)

def grad_L(theta, a):
    a_bar = sum(a) / len(a)
    return 2 * theta - 2 * a_bar

a = [-1, 1, 0, 2]
alpha = 0.1
theta0 = 3
iterations = 3

theta_values = [theta0]

for n in range(iterations):
    theta_n = theta_values[-1]
    grad = grad_L(theta_n, a)
    theta_next = theta_n - alpha * grad
    print(
        f"Iteration {n}: "
        f"theta_n = {theta_n:.4f}, "
        f"grad = {grad:.4f}, "
        f"theta_(n+1) = {theta_next:.4f}"
    )
    theta_values.append(theta_next)
```

Exercise 3

We have the same loss function and dataset as Exercise 2:

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{n}(\theta - a_i)^2, \quad a = [-1, 1, 0, 2]$$

We use **Stochastic Gradient Descent (SGD)**.

Learning rate: $\alpha = 0.2$

Starting point: $\theta_0 = 3$

Selection: At each step, we select **one random point** a_i from the dataset.

For this exercise, we assume the random selection order is: $a^{(0)} = -1$, $a^{(1)} = 1$, and $a^{(2)} = 2$

- Compute the first **three iterations**.
- Where does it converge to?

Solution

- Gradient of the single-point loss ($l_i(\theta) = (\theta - a_i)^2$). Taking the derivative w.r.t θ :

$$\nabla l_i(\theta) = 2(\theta - a_i)$$

The SGD update rule is $\theta_{n+1} = \theta_n - \alpha \nabla l_i(\theta_n) = \theta_n - \alpha \cdot 2(\theta_n - a_i)$.

- Now we can start with the iterations (starting from $\theta_0 = 3$):

Iteration 0 (Random point $a_i = -1$):

$$\nabla l(3) = 2(3 - (-1)) = 8$$
$$\theta_1 = 3 - 0.2 \cdot 8 = 1.4$$

Iteration 1 (Random point $a_i = 1$):

$$\nabla l(1.4) = 2(1.4 - 1) = 0.8$$
$$\theta_2 = 1.4 - 0.2 \cdot 0.8 = 1.24$$

Continuing with the next iterations:

Iteration	Random a_i	θ_n	$\nabla l(\theta_n)$	Update Calculation	θ_{n+1}
0	-1	3.00	8.00	$3 - 0.2 \cdot 8$	1.40
1	1	1.40	0.80	$1.4 - 0.2 \cdot 0.8$	1.24
2	2	1.24	-1.52	$1.24 - 0.2 \cdot (-1.52)$	1.544
3	0	1.544	3.088	$1.544 - 0.2 \cdot 3.088$	0.9264

- **Convergence**:

Unlike Full-Batch Gradient Descent, SGD is noisy.
Notice that in Iteration 2, the value increased from 1.24 to 1.544. This happened because the specific random point chosen (2) was an "outlier" relative to the current estimate, pulling the parameter temporarily away from the center.

However, over many iterations, the values will oscillate around and eventually converge toward the mean $\bar{a} = 0.5$.

Python code

```python
import random

def grad_stochastic(theta, ai):
    # Gradient for a single data point
    return 2 * (theta - ai)

a = [-1, 1, 0, 2]
alpha = 0.2
theta0 = 3
iterations = 3

# For reproducing the specific exercise results, we force this sequence.
# In a real scenario, you would use: random_selection = [random.choice(a) for _ in range(iterations)]
random_selection = [-1, 1, 2]

theta_values = [theta0]

for n in range(iterations):
    theta_n = theta_values[-1]
    ai = random_selection[n] # Pick the specific random point

    grad = grad_stochastic(theta_n, ai)
    theta_next = theta_n - alpha * grad

    print(
        f"Iteration {n} (using a_i={ai}): "
        f"theta_n = {theta_n:.4f}, "
        f"grad = {grad:.4f}, "
        f"theta_(n+1) = {theta_next:.4f}"
    )
    theta_values.append(theta_next)
```

Exercise 4

We have the same loss function and dataset as before:

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{n}(\theta - a_i)^2, \quad a = [-1, 1, 0, 2]$$

We use **Mini-Batch Gradient Descent**.

- **Learning rate:** $\alpha = 0.2$
- **Starting point:** $\theta_0 = 3$
- **Batch Size:** 2 *(We use 2 data points per iteration).*

For this exercise, we assume the random batch selection order is:

- Iteration 0: Batch $B_0 = \{-1, 1\}$
- Iteration 1: Batch $B_1 = \{0, 2\}$
- Compute the first two iterations.
- Where does it converge to?

Solution

Gradient for a mini-batch of size is the average of the gradients for the points in that batch:

$$\nabla L_B(\theta) = \frac{1}{m} \sum_{a_i \in B} 2(\theta - a_i)$$

The **Mini-Batch update rule** is $\theta_{n+1} = \theta_n - \alpha \nabla L_B(\theta_n)$.

- Now we can start with the iterations (starting from $\theta_0 = 3$):

Iteration 0 (Batch $B_0 = \{-1, 1\}$):

First, compute the gradient for each point in the batch:

- For $a = -1: \; 2(3 - (-1)) = 8$
- For $a = 1: \; 2(3 - 1) = 4$

Average Gradient $\nabla L_{B_0}(3)$:

$$\frac{1}{2}(8 + 4) = 6$$

Update:

$$\theta_1 = 3 - 0.2 \cdot 6 = 1.8$$

Iteration 1 (Batch $B_1 = \{0, 2\}$):

Compute the gradient for each point in the batch using the new $\theta_1 = 1.8$:

- For $a = 0: \; 2(1.8 - 0) = 3.6$

- For $a = 2: \ 2(1.8 - 2) = -0.4$

Average Gradient $\nabla L_{B_1}(1.8)$:

$$\frac{1}{2}(3.6 + (-0.4)) = 1.6$$

Update:

$$\theta_2 = 1.8 - 0.2 \cdot 1.6 = 1.48$$

Summary Table:

Iteration	Batch B_n	θ_n	Gradients in Batch	Avg Gradient ∇L_B	Update Calculation	θ_{n+1}
0	$\{-1, 1\}$	3.00	$8, 4$	6.00	$3 - 0.2 \cdot 6$	1.80
1	$\{0, 2\}$	1.80	$3.6, -0.4$	1.60	$1.8 - 0.2 \cdot 1.6$	1.48
2	...	...	...	...	...	...

- **Convergence**:

Mini-batch gradient descent is a compromise between SGD and Full-Batch. It is more stable than SGD (less noise) but faster to compute per step than Full-Batch.
The value is converging toward the mean $\bar{a} = 0.5$.

Python code

```python
import numpy as np

def grad_minibatch(theta, batch):
    # Calculate gradient for every point in the batch
    gradients = [2 * (theta - ai) for ai in batch]
    # Return the average of these gradients
    return sum(gradients) / len(batch)

a = [-1, 1, 0, 2]
alpha = 0.2
theta0 = 3
iterations = 2

# For reproducing the specific exercise results, we force these batches.
# In a real scenario, you would shuffle 'a' and slice it into chunks.
batches = [
    [-1, 1], # Iteration 0
    [0, 2]   # Iteration 1
]

theta_values = [theta0]

for n in range(iterations):
    theta_n = theta_values[-1]
    current_batch = batches[n]

    grad = grad_minibatch(theta_n, current_batch)
    theta_next = theta_n - alpha * grad

    print(
        f"Iteration {n} (Batch={current_batch}): "
        f"theta_n = {theta_n:.4f}, "
        f"Avg Grad = {grad:.4f}, "
        f"theta_(n+1) = {theta_next:.4f}"
    )
    theta_values.append(theta_next)
```

Exercise 5

We have a **multi-variable** function:

$$f(x, y) = x^2 + 2y^2 + xy$$

We use **Full-Batch Gradient Descent** with:

- **Learning rate:** $\alpha = 0.1$

- **Starting point:** $(x_0, y_0) = (3, 2)$

Compute the first **three iterations** and determine where it converges.

Solution

- **Gradient calculation:**

For a multi-variable function, we compute the **partial derivatives** with respect to each variable:

$$\frac{\partial f}{\partial x} = 2x + y$$

$$\frac{\partial f}{\partial y} = 4y + x$$

The **gradient vector** is:

$$\nabla f(x, y) = [\begin{smallmatrix} 2x + y \\ 4y + x \end{smallmatrix}]$$

Remember that the **Gradient Descent update rule** for multiple variables is:

$$[\begin{smallmatrix} x_{n+1} \\ y_{n+1} \end{smallmatrix}] = [\begin{smallmatrix} x_n \\ y_n \end{smallmatrix}] - \alpha \nabla f(x_n, y_n)$$

Or equivalently:

- $x_{n+1} = x_n - \alpha \cdot \frac{\partial f}{\partial x}$
- $y_{n+1} = y_n - \alpha \cdot \frac{\partial f}{\partial y}$
- Now we can start with the iterations (starting from $(x_0, y_0) = (3, 2)$):

Iteration 0:

$$\frac{\partial f}{\partial x}(3, 2) = 2(3) + 2 = 8$$

$$\frac{\partial f}{\partial y}(3, 2) = 4(2) + 3 = 11$$

Updates:

$$x_1 = 3 - 0.1 \cdot 8 = 2.2$$
$$y_1 = 2 - 0.1 \cdot 11 = 0.9$$

Iteration 1:

$$\frac{\partial f}{\partial x}(2.2, 0.9) = 2(2.2) + 0.9 = 5.3$$

$$\frac{\partial f}{\partial y}(2.2, 0.9) = 4(0.9) + 2.2 = 5.8$$

Updates:

$$x_2 = 2.2 - 0.1 \cdot 5.3 = 1.67$$
$$y_2 = 0.9 - 0.1 \cdot 5.8 = 0.32$$

Continuing with the next iterations:

Iteration	x_n	y_n	$\frac{\partial f}{\partial x}$	$\frac{\partial f}{\partial y}$	Update Calculation	x_{n+1}	y_{n+1}
0	3.00	2.00	8.00	11.00	$x - 0.1(8), y - 0.1(11)$	2.20	0.90
1	2.20	0.90	5.30	5.80	$x - 0.1(5.3), y - 0.1(5.8)$	1.67	0.32
2	1.67	0.32	3.66	2.95	$x - 0.1(3.66), y - 0.1(2.95)$	1.304	0.025

- **Convergence**:

To find the minimum, we set both partial derivatives to zero:

$$\frac{\partial f}{\partial x} = 2x + y = 0$$
$$\frac{\partial f}{\partial y} = 4y + x = 0$$

From the first equation: $y = -2x$

Substituting into the second: $4(-2x) + x = -8x + x = -7x = 0 \Rightarrow x = 0$

Therefore: $y = 0$

The function is **convex** (we can verify by checking the Hessian matrix is positive definite), and the minimum is achieved at:

$$(x^*, y^*) = (0, 0)$$

The iterations are clearly **converging toward** $(0, 0)$.

Note: In multi-variable optimization, all parameters are updated **simultaneously** using their respective gradients. This is the foundation of training machine learning models where we have many weights to optimize.

Python code

```python
import numpy as np

def f(x, y):
    """Objective function"""
    return x**2 + 2*y**2 + x*y

def grad_f(x, y):
    """Gradient vector: [∂f/∂x, ∂f/∂y]"""
    df_dx = 2*x + y
    df_dy = 4*y + x
    return np.array([df_dx, df_dy])

# Initial parameters
alpha = 0.1  # learning rate
x0, y0 = 3, 2  # starting point
iterations = 3

# Store trajectory
params = np.array([[x0, y0]])

print(f"Starting point: (x, y) = ({x0}, {y0})\n")

for n in range(iterations):
    x_n, y_n = params[-1]

    # Compute gradient
    grad = grad_f(x_n, y_n)

    # Update both parameters simultaneously
    x_next = x_n - alpha * grad[0]
    y_next = y_n - alpha * grad[1]

    params = np.vstack([params, [x_next, y_next]])

    print(f"Iteration {n}:")
    print(f"  Current: (x, y) = ({x_n:.4f}, {y_n:.4f})")
    print(f"  Gradient: (∂f/∂x, ∂f/∂y) = ({grad[0]:.4f}, {grad[1]:.4f})")
    print(f"  Next: (x, y) = ({x_next:.4f}, {y_next:.4f})")
    print(f"  f(x, y) = {f(x_n, y_n):.4f}\n")

print(f"Final point after {iterations} iterations: ({params[-1][0]:.4f}, {params[-1][1]:.4f})")
print(f"Converging toward the minimum at (0, 0)")
```