Lab: Gradient Descent from Scratch

COMP 395 – Deep Learning

```
pip install numpy matplotlib
```

Review: What Does the Gradient Tell Us?

Recall from last class:

- The gradient ∇f at a point is a vector of all partial derivatives
- ∇f points in the direction of **steepest increase**
- $-\nabla f$ points in the direction of **steepest decrease**
- At a minimum, $\nabla f = \mathbf{0}$

Key insight: If we want to find a minimum, we can repeatedly take small steps in the $-\nabla f$ direction until we get there.

The Gradient Descent Algorithm

Goal: Find a point (x, y) that minimizes $f(x, y)$

Algorithm:

1. Start at some initial guess (x_0, y_0)
2. Compute the gradient ∇f at the current point
3. Take a small step in the *opposite* direction of the gradient:

$$\begin{pmatrix} x_{\text{new}} \\ y_{\text{new}} \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} - \alpha \cdot \nabla f$$

4. Repeat until you reach a stopping condition

The parameter α is called the **learning rate** — it controls how big each step is.

Chain Rule Refresher: One Variable

The problem: How do we differentiate a composition of functions?

Given $h(x) = (x-3)^2$, this is really $h(x) = g(u)$ where:

- $g(u) = u^2$ (outer function)
- $u(x) = x - 3$ (inner function)

Chain Rule (function notation):

$$h'(x) = g'(u) \cdot u'(x)$$

Chain Rule (Leibniz notation):

$$\frac{dh}{dx} = \frac{dg}{du} \cdot \frac{du}{dx}$$

"Derivative of the outside times derivative of the inside."

Chain Rule Example: One Variable

Find $\frac{d}{dx}\left[(x-3)^2\right]$

Step 1: Identify outer and inner functions

- Outer: $g(u) = u^2 \implies g'(u) = 2u$
- Inner: $u = x - 3 \implies \frac{du}{dx} = 1$

Step 2: Apply chain rule

$$\frac{d}{dx}\left[(x-3)^2\right] = \underbrace{2u}_{\text{outer derivative}} \cdot \underbrace{1}_{\text{inner derivative}} = 2(x-3)$$

Another example: $\frac{d}{dx}\left[\sin(3x)\right] = \cos(3x) \cdot 3 = 3\cos(3x)$

Chain Rule with Partial Derivatives

The chain rule works the same way with partial derivatives.

Example: Let $f(x, y) = (x - 3)^2 + 2(y - 1)^2$

Find $\frac{\partial f}{\partial x}$: (treat y as constant)

The term $2(y - 1)^2$ is just a constant, so its derivative is 0.
For $(x - 3)^2$:

- Outer: $u^2 \implies$ derivative is $2u$
- Inner: $u = x - 3 \implies \frac{\partial u}{\partial x} = 1$

$$\frac{\partial f}{\partial x} = 2(x - 3) \cdot 1 + 0 = 2(x - 3)$$

Chain Rule with Partial Derivatives (continued)

Find $\frac{\partial f}{\partial y}$: (treat x as constant)

The term $(x-3)^2$ is just a constant, so its derivative is 0.
For $2(y-1)^2$:

- Outer: $2u^2 \implies$ derivative is $4u$
- Inner: $u = y - 1 \implies \frac{\partial u}{\partial y} = 1$

$$\frac{\partial f}{\partial y} = 0 + 4(y-1) \cdot 1 = 4(y-1)$$

The gradient:

$$\nabla f = \begin{pmatrix} 2(x-3) \\ 4(y-1) \end{pmatrix}$$

Today's Function

We will minimize:

$$f(x, y) = (x - 3)^2 + 2(y - 1)^2$$

Your turn: Verify the gradient we just computed.

On paper, confirm:

$$\nabla f = \begin{pmatrix} 2(x - 3) \\ 4(y - 1) \end{pmatrix}$$

Question: Where is $\nabla f = \mathbf{0}$? What does this point represent?

Check with a neighbor, then we'll move to code.

Step 1: Translate to Python

Define the function and its gradient:

```
import numpy as np
import matplotlib.pyplot as plt

def f(x, y):
    return (x - 3)**2 + 2*(y - 1)**2

def gradient_f(x, y):
    df_dx = _____________  # Fill in from your hand calculation
    df_dy = _____________  # Fill in from your hand calculation
    return np.array([df_dx, df_dy])
```

Test your gradient: What should `gradient_f(3, 1)` return?

Step 2: The Update Rule

Each iteration, we update our position:

$$\begin{pmatrix} x_{\text{new}} \\ y_{\text{new}} \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} - \alpha \cdot \nabla f(x, y)$$

In Python, if `pos` is a numpy array `[x, y]`:

```
pos = pos - alpha * gradient_f(pos[0], pos[1])
```

What happens if α is too large? Too small?

Step 3: The Loop Structure

```
# Hyperparameters
alpha = 0.1           # Learning rate
max_iters = 1000      # Maximum iterations
tolerance = 1e-6      # Stopping threshold

# Initialize
pos = np.array([0.0, 0.0])  # Starting point
history = [pos.copy()]       # Track path for plotting

for i in range(max_iters):
    grad = ______________________  # Compute gradient at current pos

    # Check stopping condition: stop if gradient is very small
    if ______________________ < tolerance:
        print(f'Converged at iteration {i}')
        break
```

Stopping Condition

When should we stop?

At a minimum, $\nabla f = \mathbf{0}$. In practice, we stop when $\|\nabla f\|$ is "small enough."

The **magnitude** (norm) of the gradient:

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

In numpy: `np.linalg.norm(grad)`

Our stopping condition:

```
if np.linalg.norm(grad) < tolerance:
```

We also cap at `max_iters` in case something goes wrong.

Visualization (Boilerplate Provided)

Copy this code to visualize your descent path:

```
# Create contour plot
x_range = np.linspace(-1, 6, 100)
y_range = np.linspace(-2, 4, 100)
X, Y = np.meshgrid(x_range, y_range)
Z = f(X, Y)

plt.figure(figsize=(10, 8))
plt.contour(X, Y, Z, levels=30, cmap='viridis')
plt.colorbar(label='f(x, y)')

# Plot the path
history = np.array(history)
plt.plot(history[:, 0], history[:, 1], 'ro-', markersize=3)
plt.plot(history[0, 0], history[0, 1], 'go', markersize=10,
         label='Start')
plt.plot(history[-1, 0], history[-1, 1], 'r*', markersize=15,
```

Experiments

Once your code is working:

1. **Try different starting points:**
   - `pos = np.array([0.0, 0.0])`
   - `pos = np.array([-5.0, 5.0])`
   - `pos = np.array([10.0, -3.0])`

   Do they all converge to the same minimum?

2. **Try different learning rates:**
   - `alpha = 0.01` (small)
   - `alpha = 0.1` (medium)
   - `alpha = 0.5` (large)
   - `alpha = 1.0` (too large?)

   What happens to the path? The number of iterations?

Checkpoint

Before you leave, verify:

1. Your code finds the minimum at approximately $(3, 1)$
2. The function value at the minimum is approximately 0
3. Your visualization shows the path descending toward the minimum
4. You can explain what happens when α is too large

Looking ahead: In neural networks, f will be a loss function and (x, y) will be thousands or millions of weights. The algorithm is the same — just more dimensions!

Complete Template

```
import numpy as np
import matplotlib.pyplot as plt

def f(x, y):
    return (x - 3)**2 + 2*(y - 1)**2

def gradient_f(x, y):
    df_dx = _______________
    df_dy = _______________
    return np.array([df_dx, df_dy])

alpha = 0.1
max_iters = 1000
tolerance = 1e-6
pos = np.array([0.0, 0.0])
history = [pos.copy()]

for i in range(max_iters):
```