Tiki’s implementation of Adadelta Optimizer

Deep Learning

I implemented the Adadelta optimizer update step function. Your function should take the current parameter value, gradient, and moving averages as inputs, and return the updated parameter value and new moving averages. The function should handle both scalar and array inputs, and include proper input validation.

Example:

Input:

```
parameter = 1.0, grad = 0.1, u = 1.0, v = 1.0, rho =
0.95, epsilon = 1e-6
```

Output:

```
(0.89743, 0.9505, 0.95053)
```

```
import numpy as np
def adadelta_optimizer(parameter, grad, u, v, rho=0.95, epsilon=1e-6):
    """
    Update parameters using the AdaDelta optimizer.
    AdaDelta is an extension of AdaGrad that seeks to reduce its
aggressive,
    monotonically decreasing learning rate.
    Args:
        parameter: Current parameter value
        grad: Current gradient
        u: Running average of squared gradients
        v: Running average of squared parameter updates
        rho: Decay rate for the moving average (default=0.95)
        epsilon: Small constant for numerical stability (default=1e-6)
    Returns:
        tuple: (updated_parameter, updated_u, updated_v)
    """
    # Your code here
```

```
  #1. Update running average of squared gradients
    u = rho * u + (1 - rho) * (grad ** 2)
    u = rho * u + (1- rho) * (grad ** 2)
    u = rho * u + ( 1 - rho) * (grad **2)
    u = rho * u  + (1 - rho) * (grad ** 2)
    u = rho * u + (1 - rho) * (grad **2)


   #2. Compute parameter update (Delta theta)
     delta = - (np.sqrt(v + epsilon) / np.sqrt(u + epsilon)) * grad
     delta = - (np.sqrt(v + epsilon) / np.sqrt(u + epsilon)) * grad
     delta = - (np.sqrt(v + epsilon) / np.sqrt(u + epsilon)) * grad
     delta = - (np.sqrt(v + epsilon) / np.sqrt(u + epsilon)) * grad
     delta = - (np.sqrt(v + epsilon) / np.sqrt(u + epsilon)) * grad

   #3. Update running average of squared parameter updates, update running average of
squared parameters
     v = rho * v + (1 - rho) * (delta ** 2)
     v = rho * v + (1 - rho) * (delta ** 2)
     v = rho * v + (1 - rho) * (delta ** 2)
     v = rho * v + (1 - rho) * (delta ** 2)
     v = rho * v + (1 - rho) * (delta ** 2)



   #4. Apply update to parameter, apply update to my parameter
     parameter = parameter + delta
     parameter = parameter + delta
     parameter = parameter + delta
     parameter = parameter + delta
     parameter = parameter + delta

   #5. Return nicely rounded values
     return np.round(parameter, 5), np.round(u, 5), np.round(v, 5)
```