Copilot

Sure, here's the equivalent code using NumPy with the original initializations and size comments:

```
import numpy as np

# Define the sizes
input_size_case1 = (1, 11088)
input_size_case2 = (16, 864)
r1_size = (32,)
r2_size = (128,)
U1_size_case1 = (1, 11088, 32)
U1_size_case2 = (16, 864, 32)
U2_size = (32, 128)

# Initialize the tensors
input_case1 = np.random.normal(size=input_size_case1)  # Size: (1, 11088)
input_case2 = np.random.normal(size=input_size_case2)  # Size: (16, 864)
r1 = np.random.normal(size=r1_size)  # Size: (32,)
r2 = np.random.normal(size=r2_size)  # Size: (128,)
U1_case1 = np.random.normal(size=U1_size_case1)  # Size: (1, 11088, 32)
U1_case2 = np.random.normal(size=U1_size_case2)  # Size: (16, 864, 32)
U2 = np.random.normal(size=U2_size)  # Size: (32, 128)

# Compute the product U1.tensordot(r1)
product_case1 = np.tensordot(U1_case1, r1, axes=([-1], [0]))  # Size: (1, 11088)
product_case2 = np.tensordot(U1_case2, r1, axes=([-1], [0]))  # Size: (16, 864)

# Compute the difference between the input and the product
difference_case1 = input_case1 - product_case1
difference_case2 = input_case2 - product_case2

# Compute the transpose of U1 along the last three axes
U1T_case1 = np.transpose(U1_case1, (2, 0, 1))  # Size: (32, 1, 11088)
U1T_case2 = np.transpose(U1_case2, (2, 0, 1))  # Size: (32, 16, 864)

# Compute the dot product of the difference and U1_transpose
gradient_case1 = np.tensordot(difference_case1, U1T_case1, axes=([0, 1], [1, 2]))  # Size: (32,)
gradient_case2 = np.tensordot(difference_case2, U1T_case2, axes=([0, 1], [1, 2]))  # Size: (32,)

# Update r1
r1 += gradient_case1
r1 += gradient_case2
```

This code does the same thing as your original TensorFlow code, but it uses NumPy instead. It computes an update for `r1` based on the difference between the input and a transformed version of `r1`. The transformation is done by a permuted `U1`.

I hope this helps! Let me know if you have any other questions. 😊