einsum

??????

自由索引（ ***Free indices***）和求和索引（ ***Summation indices***）：

- 自由索引，出现在箭头右边的索引，比如上面的例子就是 i 和 j；
- 求和索引，只出现在箭头左边的索引，表示中间计算结果需要这个维度上求和之后才能得到输出，比如上面的例子就是 k；

??????

- 规则一， equation 箭头左边，在不同输入之间重复出现的索引表示，把输入张量沿着该维度做乘法操作，比如还是以上面矩阵乘法为例， "ik,kj->ij"， k 在输入中重复出现，所以就是把 a 和 b 沿着 k 这个维度作相乘操作；
- 规则二，只出现在 equation 箭头左边的索引，表示中间计算结果需要在这个维度上求和，也就是上面提到的求和索引；
- 规则三， equation 箭头右边的索引顺序可以是任意的，比如上面的 "ik,kj->ij" 如果写成 "ik,kj->ji" ，那么就是返回输出结果的转置，用户只需要定义好索引的顺序，转置操作会在 einsum 内部完成。

????

特殊规则有两条：

- equation 可以不写包括箭头在内的右边部分，那么在这种情况下，输出张量的维度会根据默认规则推导。就是把输入中只出现一次的索引取出来，然后按字母表顺序排列，比如上面的矩阵乘法 "ik,kj->ij" 也可以简化为 "ik,kj"，根据默认规则，输出就是 "ij" 与原来一样；
- equation 中支持 "..." 省略号，用于表示用户并不关心的索引，比如只对一个高维张量的最后两维做转置可以这么写：

```
a = torch.randn(2,3,5,7,9)
# i = 7, j = 9
b = torch.einsum('...ij->...ji', [a])
```

?????bilinear transformation?

```
np_a = a.numpy()
np_b = b.numpy()
np_c = c.numpy()
np_out = np.empty((2, 5), dtype=np.float32)
```

```
np_out = torch.einsum('ik,jkl,il->ij', [a, b, c]).numpy()
# ik broadcast成ikl
# il broadcast成 ikl
# 'ik,jkl,il->ij'可以理解成'ikl,jkl,ikl->ij'

for i in range(0, 2):
    for j in range(0, 5):
        # 求和索引内循环 这里是 k 和 l
        sum_result = 0
        for k in range(0, 3):
            for l in range(0, 7):
                sum_result += np_a[i, k] * np_b[j, k, l] * np_c[i, l]
        np_out[i, j] = sum_result
```

??

```
a = torch.rand(2,3)
b = torch.rand(3,4)
c = torch.einsum("ik,kj->ij", [a, b])
# 等价操作 torch.mm(a, b)
```

equation

中的字符也可以理解为索引，就是输出张量的某个位置的值，是怎么从输入张量中得到的，比如上面矩阵乘法的输出 c 的某个点 c[i, j] 的值是通过 a[i, k] 和 b[k, j] 沿着 k 这个维度做内积得到的。

Revision #2
Created 2025-01-11 09:46:28 UTC by Colin
Updated 2026-04-29 07:34:51 UTC by Colin