```
def hello():
    features = []
    for i in range(3):
        features.append(torch.randn(2, 1))  # 先收集张量到列表
    # features = torch.stack(features, dim=0)  # 最后一次性堆叠
    fes = torch.stack(features, dim=1)
    return fes
```

以下是分别使用 dim=1 和 dim=2 堆叠后的结果样例（以具体数值和形状展示）：

---

1. 初始数据

假设列表 features 包含以下 3 个张量（每个形状为 (2, 1)）：

```
import torch

A = torch.tensor([[1.0], [2.0]])  # 形状 (2, 1)
B = torch.tensor([[3.0], [4.0]])  # 形状 (2, 1)
C = torch.tensor([[5.0], [6.0]])  # 形状 (2, 1)
features = [A, B, C]

```

---

2. 堆叠结果对比

(1) dim=1 的堆叠结果

```
stack_dim1 = torch.stack(features, dim=1)
```

- **输出**:

```
tensor([[[1.0],  # 第0行的3个列向量
         [3.0],
         [5.0]],

        [[2.0],  # 第1行的3个列向量
         [4.0],
         [6.0]]])
```

- **形状**：(2, 3, 1)
  - 含义：2 行 → 3 个堆叠的列 → 每列 1 个元素。

(2) dim=2 的堆叠结果

```
stack_dim2 = torch.stack(features, dim=2)
```

- **输出**：

```
tensor([[[1.0, 3.0, 5.0]],  # 第0行的1列，堆叠3个深度

        [[2.0, 4.0, 6.0]]]) # 第1行的1列，堆叠3个深度
```

- **形状**：(2, 1, 3)
  - 含义：2 行 → 1 列 → 每列 3 个堆叠的深度值。

---

3. 形状变化总结

堆叠维度	输出形状	直观解释
dim=0	(3, 2, 1)	3 个张量上下堆叠（新增“批次”维度）
dim=1	(2, 3, 1)	每行的列横向扩展（新增“特征”维度） s
dim=2	(2, 1, 3)	每列的元素纵向堆叠（新增“深度”维度）

---

4. 可视化对比

dim=1 的堆叠效果

```
原始张量:       堆叠后 (dim=1):
A = [1]        [[ [1], [3], [5] ],
     [2]         [ [2], [4], [6] ] ]
B = [3]
     [4]
C = [5]
     [6]

```

dim=2 的堆叠效果

```
原始张量:       堆叠后 (dim=2):
A = [1]        [[ [1, 3, 5] ],
     [2]         [ [2, 4, 6] ] ]
B = [3]
     [4]
C = [5]
     [6]

```

---

5. 关键结论

- **dim=1**：适合横向扩展特征（如合并多个时间步的数据）。
- **dim=2**：适合纵向堆叠通道（如合并多个传感器的读数）。
- **选择依据**：根据后续操作的需求决定堆叠维度（如卷积层通常需要 dim=1 作为通道维度）。