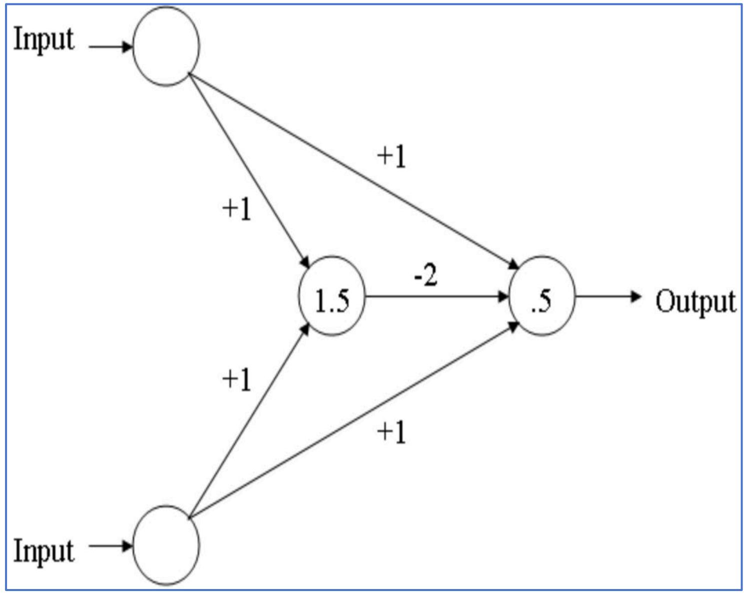
作业 3-1： 验证以下多层感知机可否实现XOR

1



Input		Output
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

作业 3-1： 该多层感知机可实现XOR

x1	x2	w11	w12	T1	f1(·)	w21	w22	w23	T2	f2(·)
0	0	1	1	1.5	G(z-T)	1	1	-2	0.5	G(z-T)
0	1	1	1	1.5	G(z-T)	1	1	-2	0.5	G(z-T)
1	0	1	1	1.5	G(z-T)	1	1	-2	0.5	G(z-T)
1	1	1	1	1.5	G(z-T)	1	1	-2	0.5	G(z-T)

z1=w11x1+w12x2	Output y1=G(z1-T1)	z2=w21x1+w22x2+x23y1	Output y2=G(z2-T2)	A XOR B
0*1+0*1 = 0	0	0*1+0*1+(-2)*0 = 0	0	0
0*1+1*1 = 1	0	0*1+1*1+(-2)*0 = 1	1	1
1*1+0*1 = 1	0	1*1+0*1+(-2)*0 = 1	1	1
1*1+1*1 = 2	1	1*1+1*1+(-2)*1 = 0	0	0

作业 3-2

1. 假设我们现在有如下数据集，其中花朵颜色和叶子形状是离散特征，花朵种类为其标签。现在，我们想通过决策树对数据集中花朵种类进行分类，请回答以下问题。

序号	花朵颜色	叶子形状	花朵种类
1	红	圆	Iris-Ame
2	红	长条	Iris-Ame
3	黑	针状	Iris-Ame
4	黑	针状	Iris-Somnus
5	黑	长条	Iris-Somnus
6	紫	圆	Iris-Somnus
7	紫	针状	Iris-XinQ
8	红	圆	Iris- XinQ
9	紫	长条	Iris- XinQ

（1）请分别计算花朵颜色和叶子形状作为选择特征的信息增益。

（2）请根据（1）问的计算结果说明应该选择哪个特征？

作业 3-2：决策树

```python
import numpy as np
import pandas as pd
```

✓ 0.0s

```python
df = pd.read_csv('HW3-2.csv')
df
```

✓ 0.0s

	花朵颜色	叶子形状	花朵种类
0	红	圆	Iris-Ame
1	红	长条	Iris-Ame
2	黑	针状	Iris-Ame
3	黑	针状	Iris-Somnus
4	黑	长条	Iris-Somnus
5	紫	圆	Iris-Somnus
6	紫	针状	Iris-XinQ
7	红	圆	Iris-XinQ
8	紫	长条	Iris-XinQ

```python
def compute_entropy(df, target):
    entropy = 0
    for i in df[target].unique():
        p = sum(df[target] == i) / len(df)
        entropy -= p * np.log2(p)
    return entropy

def gain(df, target, attribute):
    gain = compute_entropy(df, target)
    for i in df[attribute].unique():
        sub_df = df[df[attribute] == i]
        gain -= len(sub_df) / len(df) * compute_entropy(sub_df, target)
    return gain
```

```python
features = df.columns[:2]
for feature in features:
    print(feature)
    print(f"Gain: {gain(df, feature, '花朵种类'):.4f}")
```

```
花朵颜色
Gain: 0.6667
叶子形状
Gain: 0.0000
```

所以选择花朵颜色作为划分特征

作业 3-3

- 假设有以下8个点 ：
  （5, 1），（5, 2） ，（4, 1） ，（4, 2）
  （1, 3） ，（1, 4） ，（2, 3） ，（2, 4）
- 通过K-means算法进行2聚类。初始聚类中心定为（0, 4） ， （3, 3）
- 请写下详细计算步骤

作业 3-3：验证K-means

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

0.0s Python

```python
def kmeans(data: np.ndarray, n_cl: int):
    """
    K-means clustering
    :param data: np.ndarray of shape (n_samples, n_features)
    :param n_cl: number of clusters
    :return: np.ndarray of shape (n_samples,) with cluster indices
    """
    n_samples, n_features = data.shape

    # Initialize cluster centers
    # centers = data[np.random.choice(n_samples, n_cl, replace=False)]
    centers = [[0,4], [3,3]]

    # Initialize cluster indices
    clusters = np.zeros(n_samples, dtype=int)

    # Initialize distances
    distances = np.zeros((n_samples, n_cl))
```

```python
    # Main loop
    loop = 0
    while True:
        loop += 1
        print(f"Loop {loop}:")

        print(f"Centers: {centers[0]}, {centers[1]}")

        # Compute distances
        for i in range(n_cl):
            distances[:, i] = np.linalg.norm(data - centers[i], axis=1)
        print(f"Distances:\n{distances}")

        # Assign clusters
        new_clusters = np.argmin(distances, axis=1)
        print(f"new_Clusters: {new_clusters}")

        # Check for convergence
        if np.all(new_clusters == clusters):
            break
        clusters = new_clusters

        # Update cluster centers
        for i in range(n_cl):
            centers[i] = np.mean(data[clusters == i], axis=0)

        print()
    return clusters
```

作业 3-3： 验证K-means

```python
data = np.array([[5,1], [5,2], [4,1], [4,2], [1,3], [1,4], [2,3], [2,4]])
clusters = kmeans(data, 2)
print(f"Final clusters: {clusters}")
```

✓ 0.0s

```
Loop 1:
Centers: [0, 4], [3, 3]
Distances:
[[5.83095189 2.82842712]
 [5.38516481 2.23606798]
 [5.         2.23606798]
 [4.47213595 1.41421356]
 [1.41421356 2.        ]
 [1.         2.23606798]
 [2.23606798 1.        ]
 [2.         1.41421356]]
new_Clusters: [1 1 1 1 0 0 1 1]
```

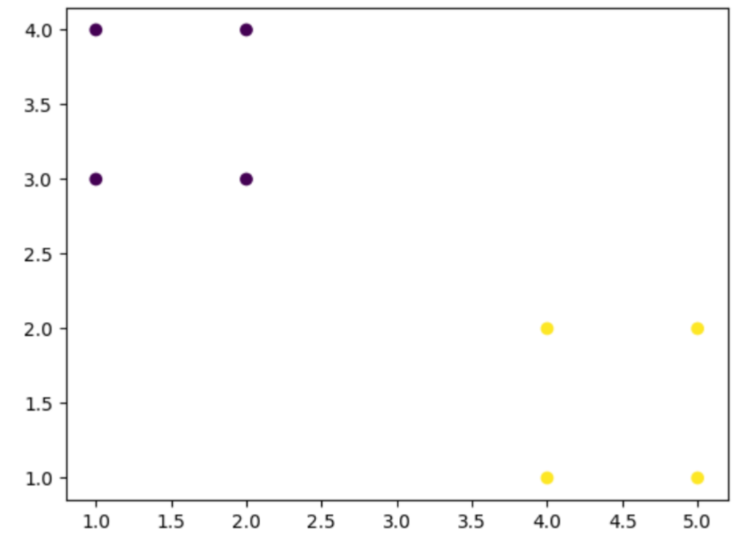
```
Loop 2:
Centers: [1.  3.5], [3.66666667 2.16666667]
Distances:
[[4.71699057 1.77169097]
 [4.27200187 1.34370962]
 [3.90512484 1.21335165]
 [3.35410197 0.372678  ]
 [0.5        2.79384244]
 [0.5        3.23608131]
 [1.11803399 1.86338998]
 [1.11803399 2.47767812]]
new_Clusters: [1 1 1 1 0 0 0 0]
```

```
Loop 3:
Centers: [1.5 3.5], [4.5 1.5]
Distances:
[[4.30116263 0.70710678]
 [3.80788655 0.70710678]
 [3.53553391 0.70710678]
 [2.91547595 0.70710678]
 [0.70710678 3.80788655]
 [0.70710678 4.30116263]
 [0.70710678 2.91547595]
 [0.70710678 3.53553391]]
new_Clusters: [1 1 1 1 0 0 0 0]

Final clusters: [1 1 1 1 0 0 0 0]
```

```python
# draw the data points
plt.scatter(data[:, 0], data[:, 1], c=clusters)
plt.show()
```

✓ 0.0s



The left column of distance is the distance from every point to center 1

The right column of distance is the distance from every point to center 2