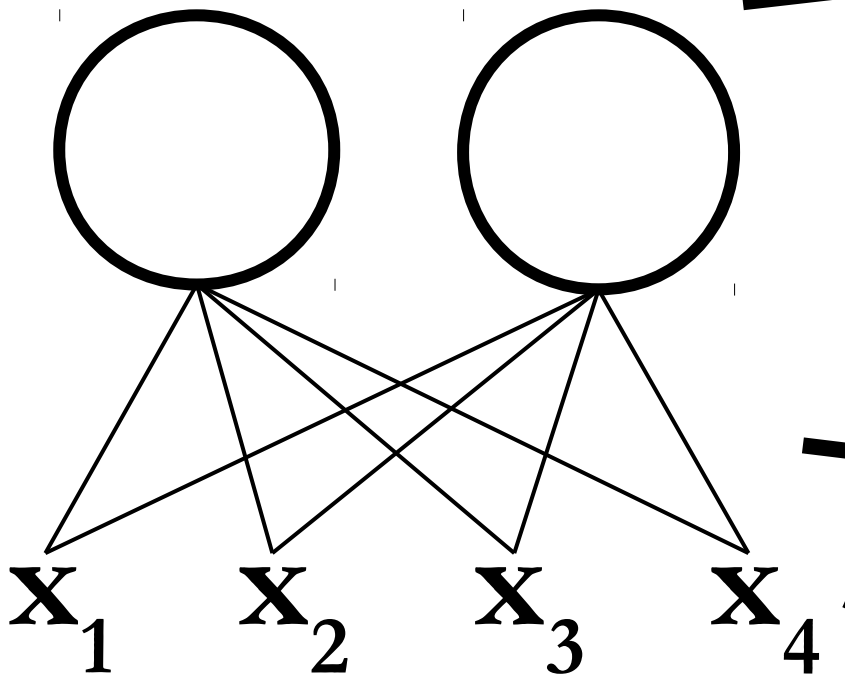
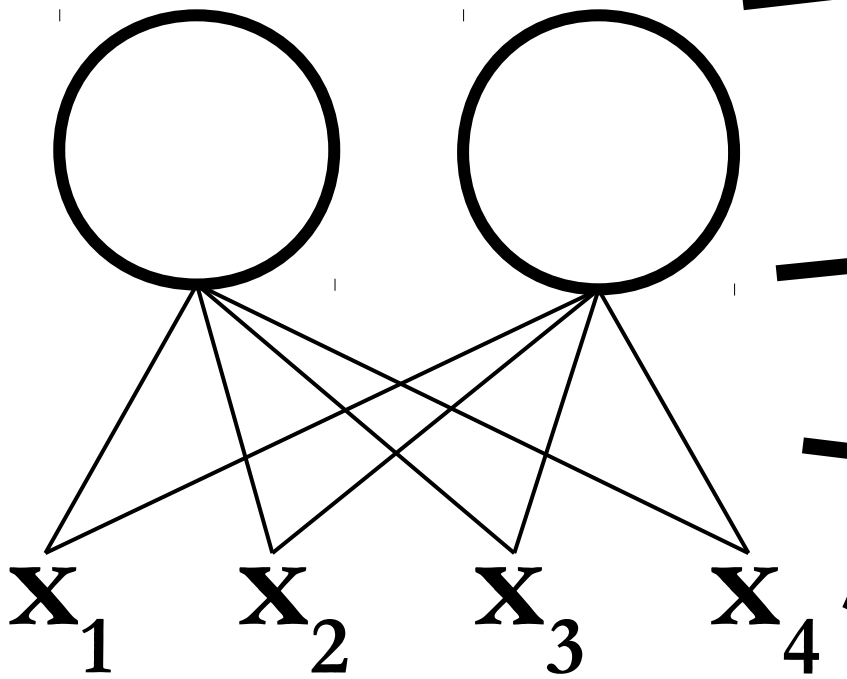



```
[[1  0]
 [1  1]
 [0  1]]
```

```
[[ 0.9   0.8  -1.0  -1.0]
 [-0.5  -0.5  1.5    1.0]]
```

```
[[4.9   3.0   1.4   0.2]
 [6.4   3.2   4.5   1.5]
 [5.8   2.7   5.1   1.9]]
```

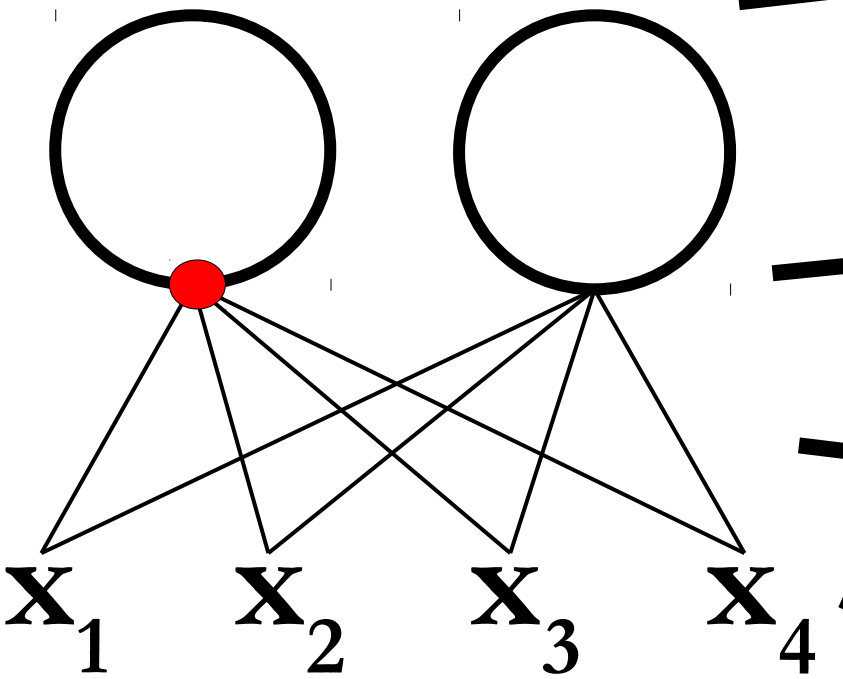



```
[[1  0]
 [1  1]
 [0  1]]
```

```
[[5.21  -1.65]
 [2.32   3.45]
 [0.38   5.30]]
```

```
[[ 0.9   0.8  -1.0  -1.0]
 [-0.5  -0.5  1.5    1.0]]
```

```
[[4.9   3.0   1.4   0.2]
 [6.4   3.2   4.5   1.5]
 [5.8   2.7   5.1   1.9]]
```

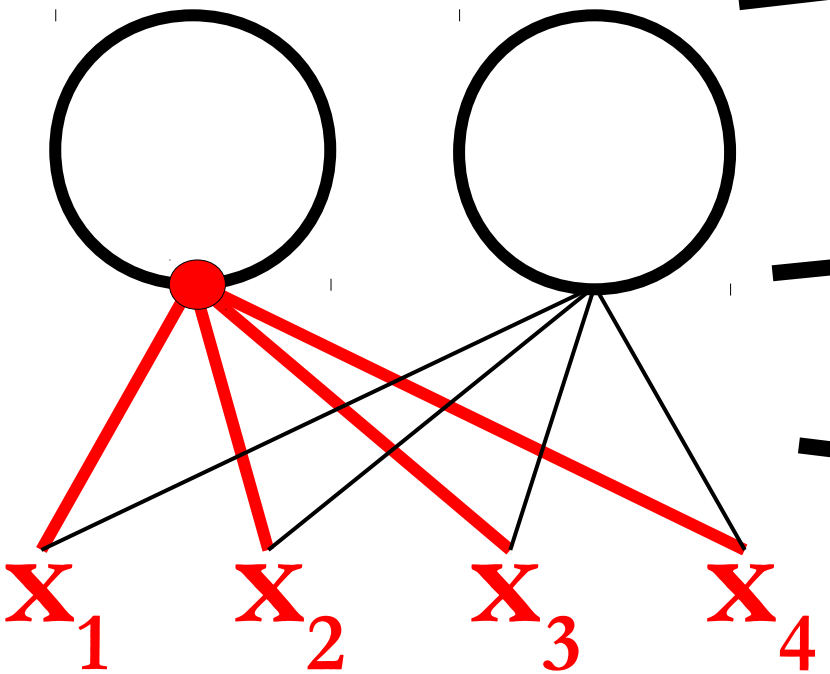



```
[[1  0]
 [1  1]
 [0  1]]
```

```
[[5.21 -1.65]
 [2.32  3.45]
 [0.38  5.30]]
```

```
[[ 0.9   0.8  -1.0  -1.0]
 [-0.5  -0.5  1.5    1.0]]
```

```
[[4.9  3.0  1.4  0.2]
 [6.4  3.2  4.5  1.5]
 [5.8  2.7  5.1  1.9]]
```

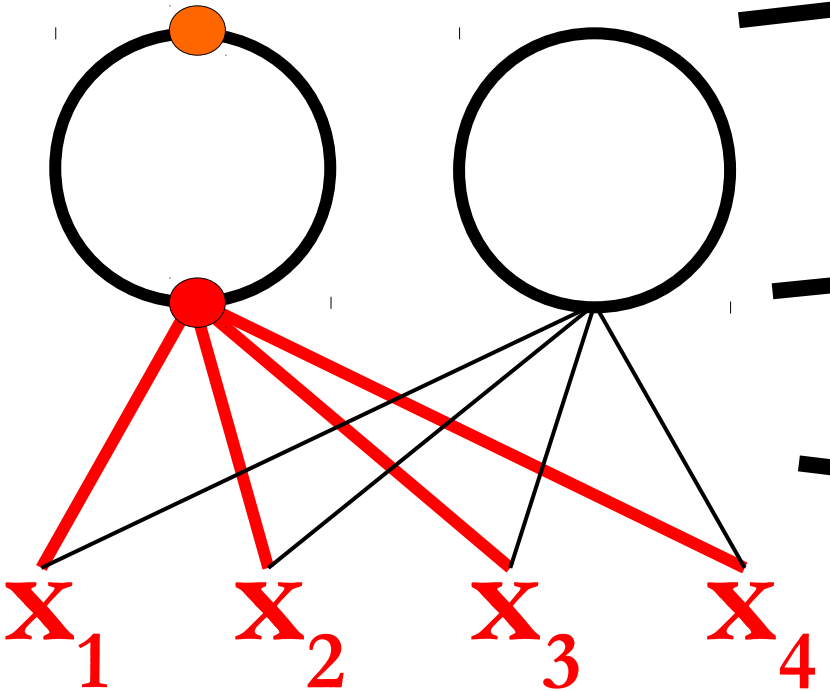



```
[[1  0]
 [1  1]
 [0  1]]
```

```
[[5.21 -1.65]
 [2.32  3.45]
 [0.38  5.30]]
```

```
[[ 0.9  0.8 -1.0 -1.0]
 [-0.5 -0.5  1.5  1.0]]
```

```
[[4.9  3.0  1.4  0.2]
 [6.4  3.2  4.5  1.5]
 [5.8  2.7  5.1  1.9]]
```

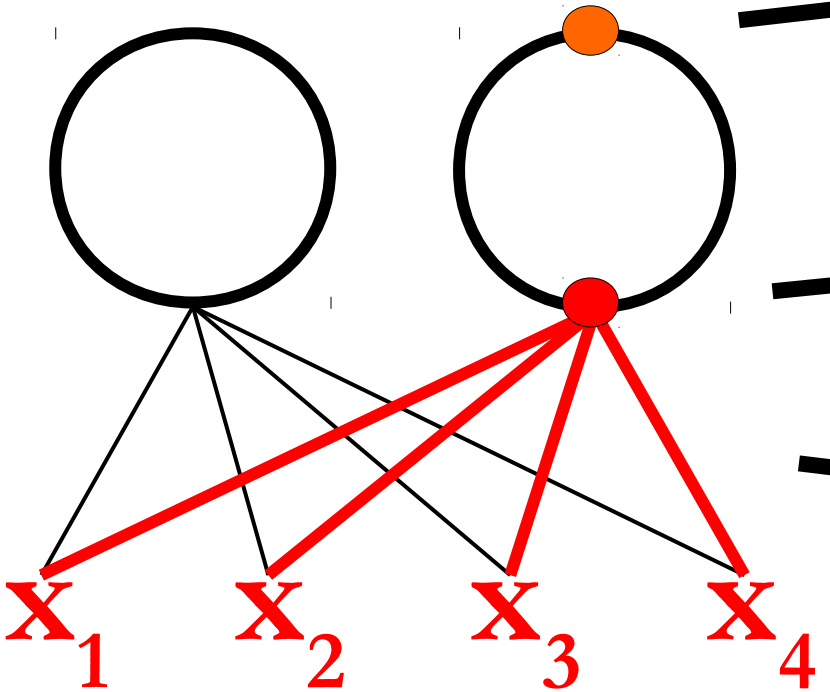


[[1 0]
[1 1]
[0 1]]

[[5.21 -1.65]
[2.32 3.45]
[0.38 5.30]]

[[0.9 0.8 -1.0 -1.0]
[-0.5 -0.5 1.5 1.0]]

[[4.9 3.0 1.4 0.2]
[6.4 3.2 4.5 1.5]
[5.8 2.7 5.1 1.9]]

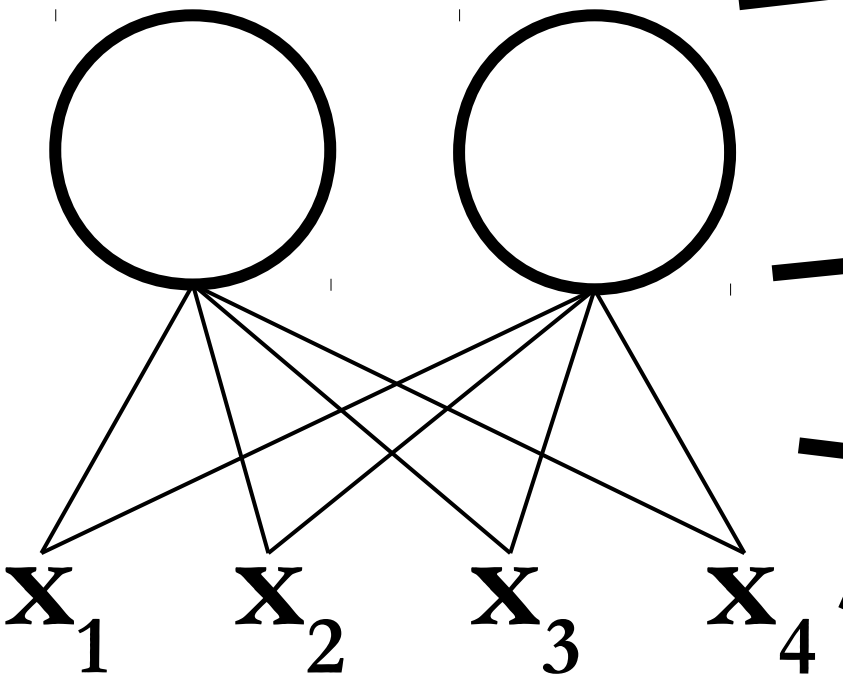



```
[[1  0]
 [1  1]
 [0  1]]
```

```
[[5.21 -1.65]
 [2.32  3.45]
 [0.38  5.30]]
```

```
[[ 0.9   0.8  -1.0  -1.0]
 [-0.5  -0.5   1.5   1.0]]
```

```
[[4.9  3.0  1.4  0.2]
 [6.4  3.2  4.5  1.5]
 [5.8  2.7  5.1  1.9]]
```

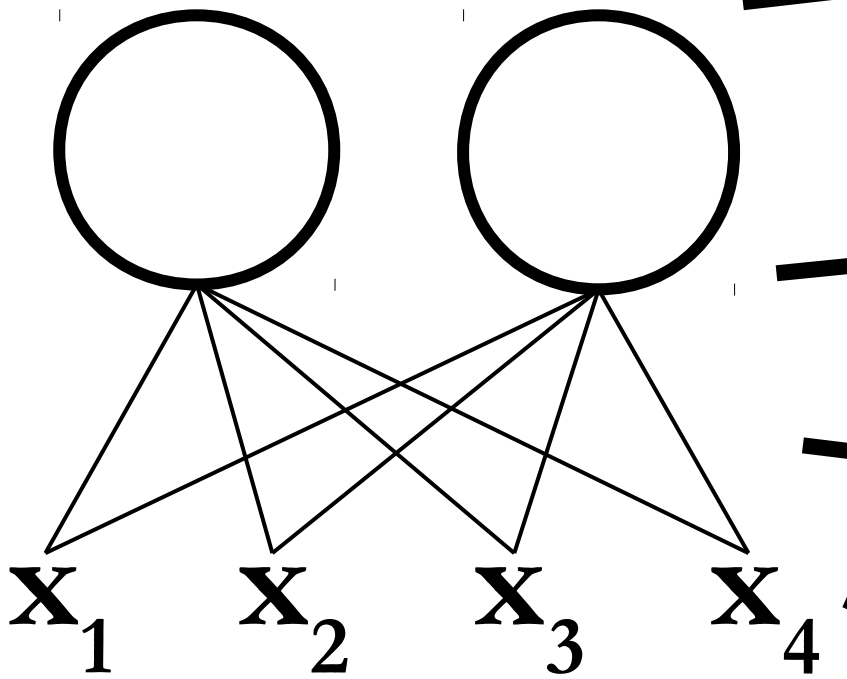



```
[[1  0]
 [1  1]
 [0  1]]
```

```
[[5.21  -1.65]
 [2.32   3.45]
 [0.38   5.30]]
```

```
[[ 0.9   0.8  -1.0  -1.0]
 [-0.5  -0.5   1.5   1.0]]
```

```
[[4.9   3.0   1.4   0.2]
 [6.4   3.2   4.5   1.5]
 [5.8   2.7   5.1   1.9]]
```

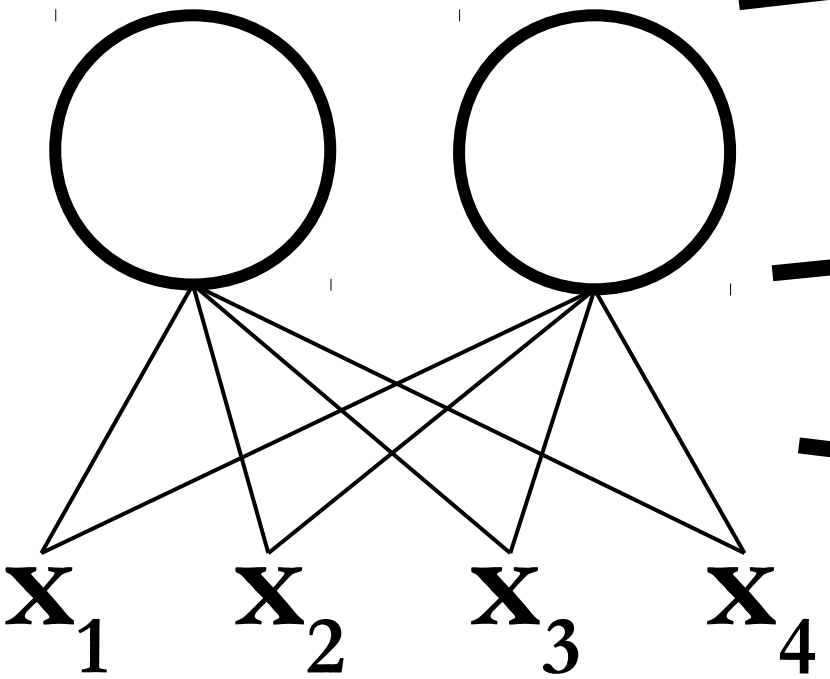



```
[[1  0]
 [1  1]
 [0  1]]
```

```
[[5.21  -1.65]
 [2.32   3.45]
 [0.38   5.30]]
```

```
[[ 0.9   0.8  -1.0  -1.0]
 [-0.5  -0.5  1.5    1.0]]
```

```
[[4.9   3.0   1.4   0.2]
 [6.4   3.2   4.5   1.5]
 [5.8   2.7   5.1   1.9]]
```

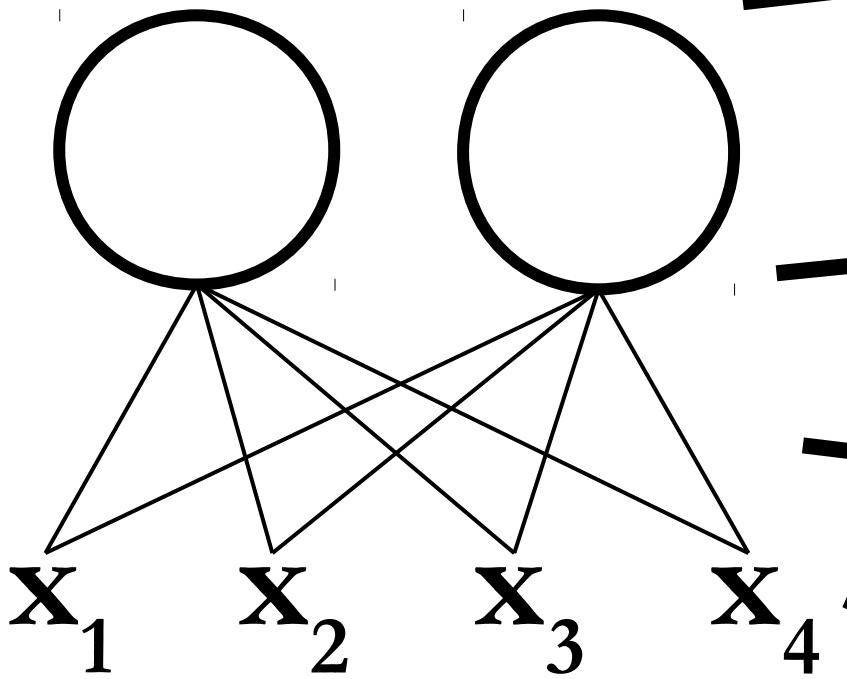



```
[[1  0]
 [1  1]
 [0  1]]
```

```
[[5.21  -1.65]
 [2.32   3.45]
 [0.38   5.30]]
```

```
[[ 0.9   0.8  -1.0  -1.0]
 [-0.5  -0.5  1.5    1.0]]
```

```
[[4.9   3.0   1.4   0.2]
 [6.4   3.2   4.5   1.5]
 [5.8   2.7   5.1   1.9]]
```

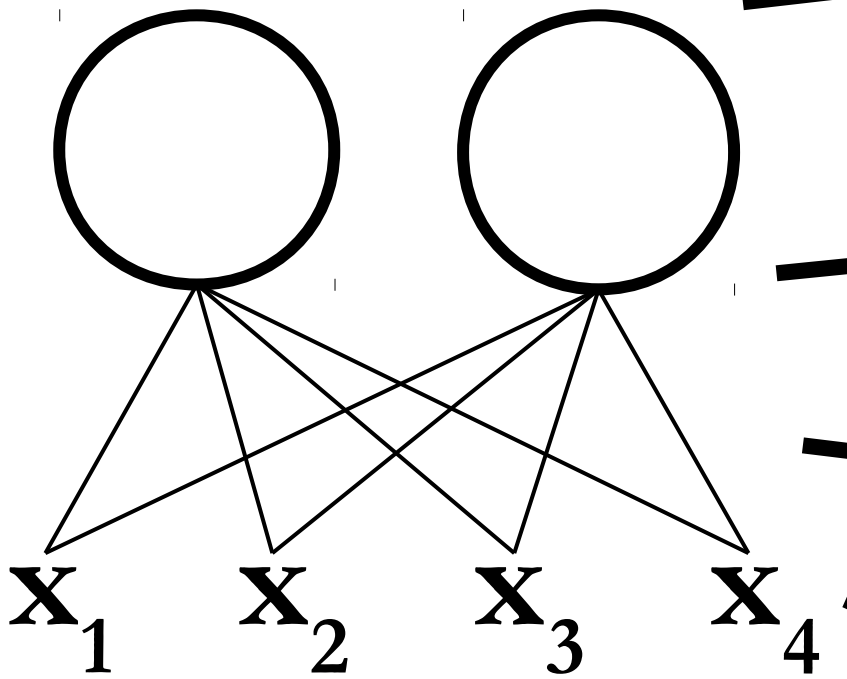



```
[[1  0]
 [1  1]
 [0  1]]
```

```
[[5.21  -1.65]
 [2.32   3.45]
 [0.38   5.30]]
```

```
[[ 0.9   0.8  -1.0  -1.0]
 [-0.5  -0.5  1.5    1.0]]
```

```
[[4.9   3.0   1.4   0.2]
 [6.4   3.2   4.5   1.5]
 [5.8   2.7   5.1   1.9]]
```




```
[[1  0]
 [1  1]
 [0  1]]
```

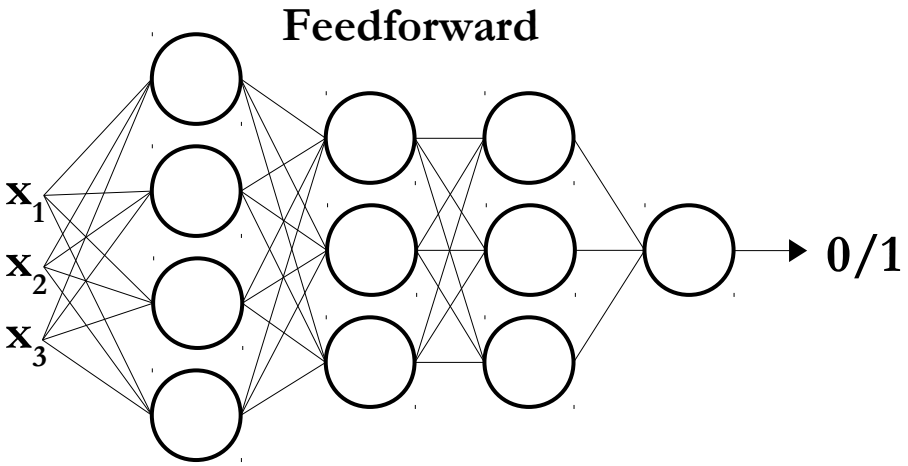
```
[[5.21  -1.65]
 [2.32   3.45]
 [0.38   5.30]]
```

```
[[ 0.9   0.8  -1.0  -1.0]
 [-0.5  -0.5  1.5    1.0]]
```

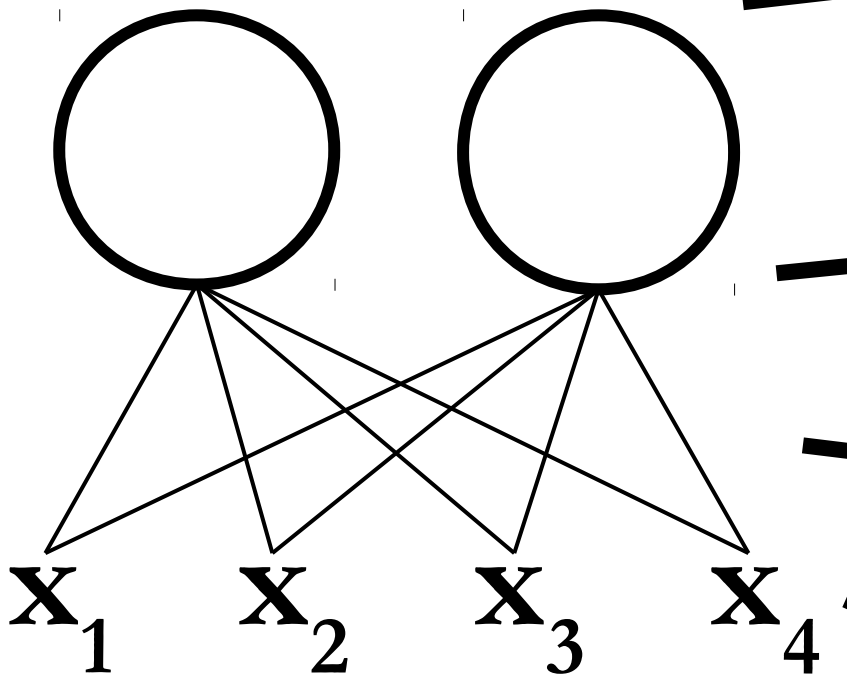
```
[[4.9   3.0   1.4   0.2]
 [6.4   3.2   4.5   1.5]
 [5.8   2.7   5.1   1.9]]
```

Deep Learning

How does the algorithm make a decision?



How do you determine the right parameters for the algorithm?



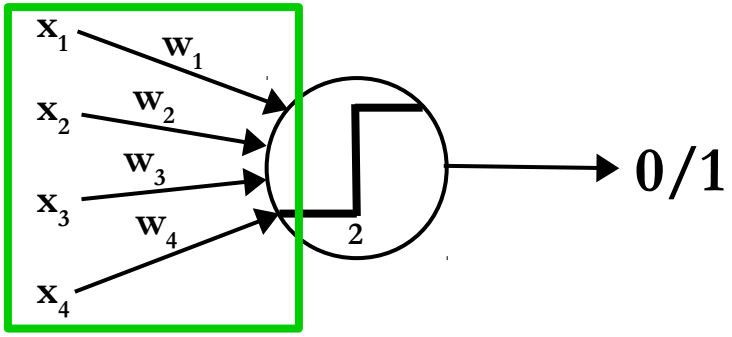

```
[[1  0]
 [1  1]
 [0  1]]
```

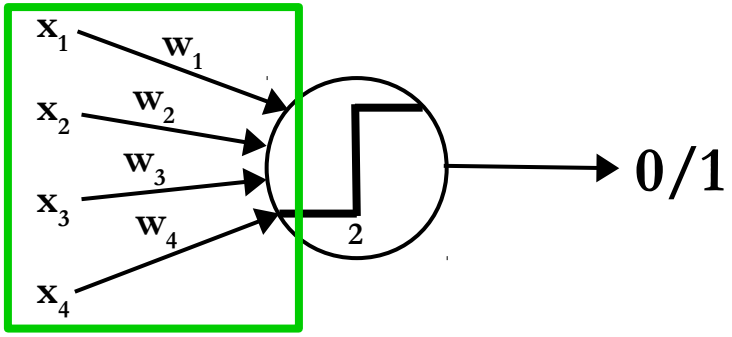
```
[[5.21  -1.65]
 [2.32   3.45]
 [0.38   5.30]]
```

```
[[ 0.9   0.8  -1.0  -1.0]
 [-0.5  -0.5  1.5    1.0]]
```

```
[[4.9   3.0   1.4   0.2]
 [6.4   3.2   4.5   1.5]
 [5.8   2.7   5.1   1.9]]
```

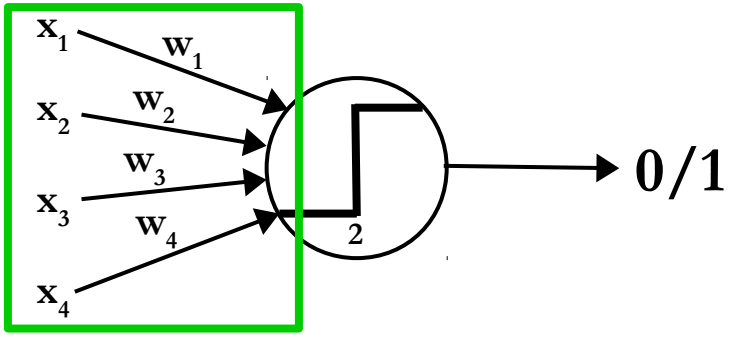
Number of examples * number of nodes * number of input values = number of iterations


x1
x2
x3
x4
w1
w2
w3
w4
2
0/1



Weighted Sum

$$\sum_{i=1}^{4} x_i w_i = x_1 w_1 + x_2 w_2 + x_3 w_3 + x_4 w_4$$

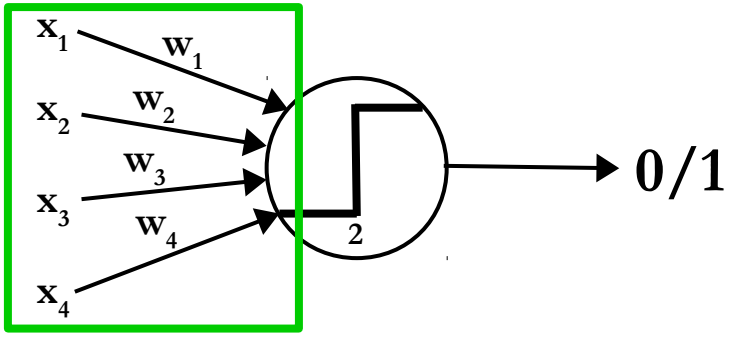


Weighted Sum

$$\sum_{i=1}^{4} x_i w_i = x_1 w_1 + x_2 w_2 + x_3 w_3 + x_4 w_4$$

Dot Product

$$\vec{x} \cdot \vec{w} = x_1 w_1 + x_2 w_2 + x_3 w_3 + x_4 w_4$$



Weighted Sum

$$\sum_{i=1}^{4} x_i w_i = x_1 w_1 + x_2 w_2 + x_3 w_3 + x_4 w_4$$

Dot Product

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{pmatrix} = x_1 w_1 + x_2 w_2 + x_3 w_3 + x_4 w_4$$

Linear Algebra

- **BLAS, LAPACK ➡ optimized algorithms**

Linear Algebra

- **BLAS, LAPACK** ➡ **optimized algorithms**

- **C, Fortran** ➡ **low-level languages**

Linear Algebra

- BLAS, LAPACK ➡ optimized algorithms

- C, Fortran ➡ low-level languages

- SIMD ➡ parallel processing

Multiplying respective Elements of two Lists

```
list_1 = [1, 2, 5]
list_2 = [4, 7, 3]
```

pure Python

```
list_3 = []
for value_1, value_2 in zip(list_1, list_2):
    value_3 = value_1 * value_2
    list_3.append(value_3)
```

```
print(list_3)
```

```
[4, 14, 15]
```

SIMD

```
list_3 = list_1 * list_2
```

```
print(list_3)
```

```
[4, 14, 15]
```

Multiplying respective Elements of two Lists

```
list_1 = [1, 2, 5]
list_2 = [4, 7, 3]
```

pure Python

```
list_3 = []
for value_1, value_2 in zip(list_1, list_2):
    value_3 = value_1 * value_2
    list_3.append(value_3)
```

```
print(list_3)
```

```
[4, 14, 15]
```

SIMD/Vectorization

```
list_3 = list_1 * list_2
```

```
print(list_3)
```

```
[4, 14, 15]
```

Linear Algebra

- BLAS, LAPACK → optimized algorithms
- C, Fortran → low-level languages
- SIMD → parallel processing

} NumPy