2022 High Energy Nuclear Physics School

Perceptron

By Hye Jin Park

hyejin.park@inha.ac.kr

Machine learning

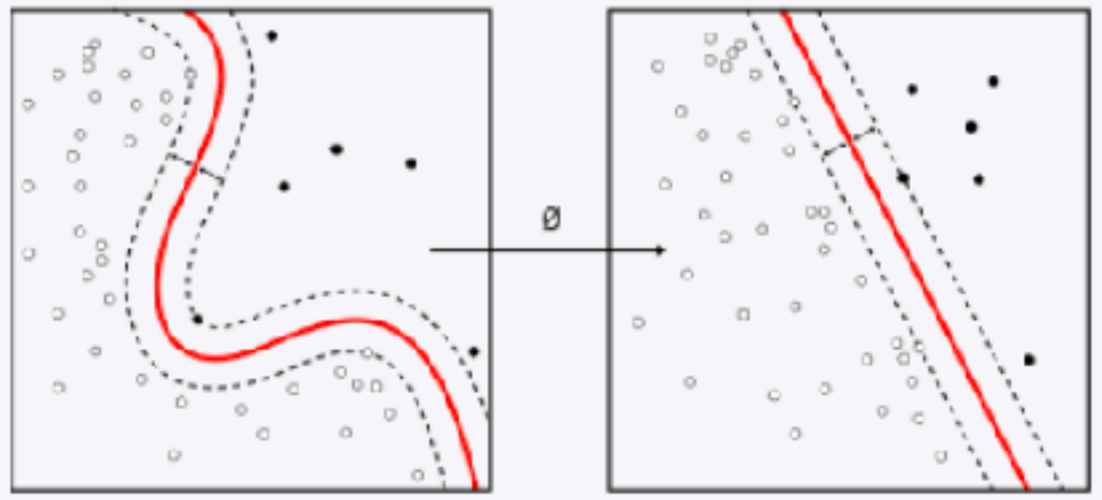
From Wikipedia, the free encyclopedia

For the journal, see Machine Learning (journal).

"Statistical learning" redirects here. For statistical learning in linguistics, see statistical learning in language acquisition.

Machine learning (**ML**) is a field of inquiry devoted to understanding and building methods that 'learn', that is, methods that leverage data to improve performance on some set of tasks.[1] It is seen as a part of artificial intelligence. Machine learning algorithms build a model based on sample data, known as training data, in order to make predictions or decisions without being explicitly programmed to do so.[2] Machine learning algorithms are used in a wide variety of applications, such as in medicine, email filtering, speech recognition, agriculture, and computer vision, where it is difficult or unfeasible to develop conventional algorithms to perform the needed tasks.[3][4] A subset of machine learning is closely related to computational statistics, which focuses on making predictions using computers, but not all machine learning is statistical learning. The study of mathematical optimization delivers methods, theory and application domains to the field of machine learning. Data mining is a related field of study, focusing on exploratory data analysis through unsupervised learning.[6][7] Some implementations of machine learning use data and neural networks in a way that mimics the working of a biological brain.[8][9] In its application across business problems, machine learning is also referred to as predictive analytics.

Part of a series on
Machine learning and data mining



Problems [show]

Supervised learning [show]
(**classification** • **regression**)

Clustering [show]

Dimensionality reduction [show]

Machine learning

From Wikipedia, the free encyclopedia

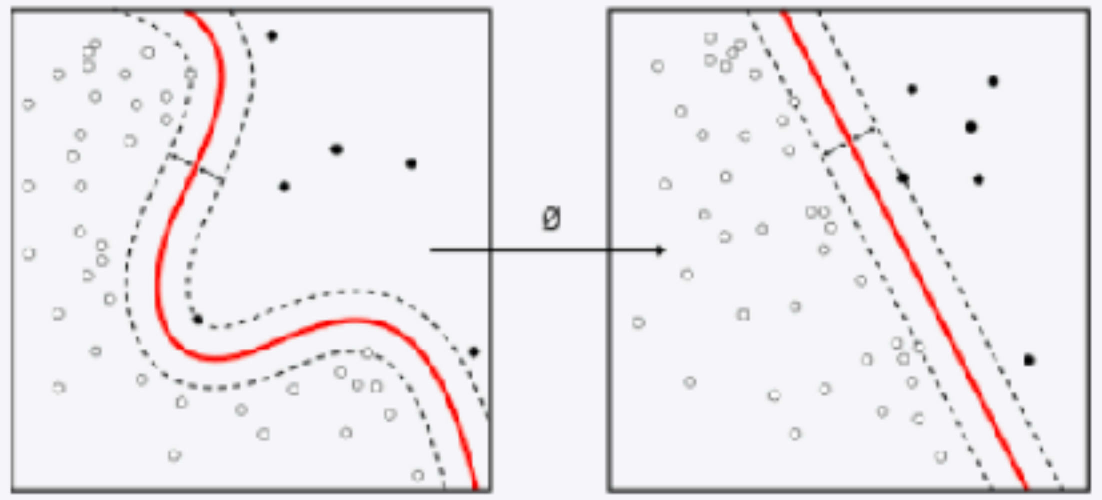
For the journal, see Machine Learning *(journal)*.

"Statistical learning" redirects here. For statistical learni *in language acquisition.*

“답을 찾아가는 방법”
“Developing methods”

Machine learning (**ML**) is a field of inquiry devoted to understanding and building methods that 'learn', that is, methods that leverage data to improve performance on some set of tasks.[1] It is seen as a part of artificial intelligence. Machine learning algorithms build a model based on sample data, known as training data, in order to make predictions or decisions without being explicitly programmed to do so.[2] Machine learning algorithms are used in a wide variety of applications, such as in medicine, email filtering, speech recognition, agriculture, and computer vision, where it is difficult or unfeasible to develop conventional algorithms to perform the needed tasks.[3][4] A subset of machine learning is closely related to computational statistics, which focuses on making predictions using computers, but not all machine learning is statistical learning. The study of mathematical optimization delivers methods, theory and application domains to the field of machine learning. Data mining is a related field of study, focusing on exploratory data analysis through unsupervised learning.[6][7] Some implementations of machine learning use data and neural networks in a way that mimics the working of a biological brain.[8][9] In its application across business problems, machine learning is also referred to as predictive analytics.

Part of a series on
Machine learning and data mining



Problems	[show]
Supervised learning (**classification** • **regression**)	[show]
Clustering	[show]
Dimensionality reduction	[show]

Supervised learning

Data with tag

Unsupervised learning

Data without tag

Supervised learning

Data with tag

	inputs			output
	Temper ature	Wind (m/s)	Rain (mm)	Play tennis
Mon.	28	1	1	Yes
Tue.	23	2	5	No
Wed.	22	8	0	No
Thur.	21	3	0	Yes
	22	1	0	?

Unsupervised learning

Data without tag

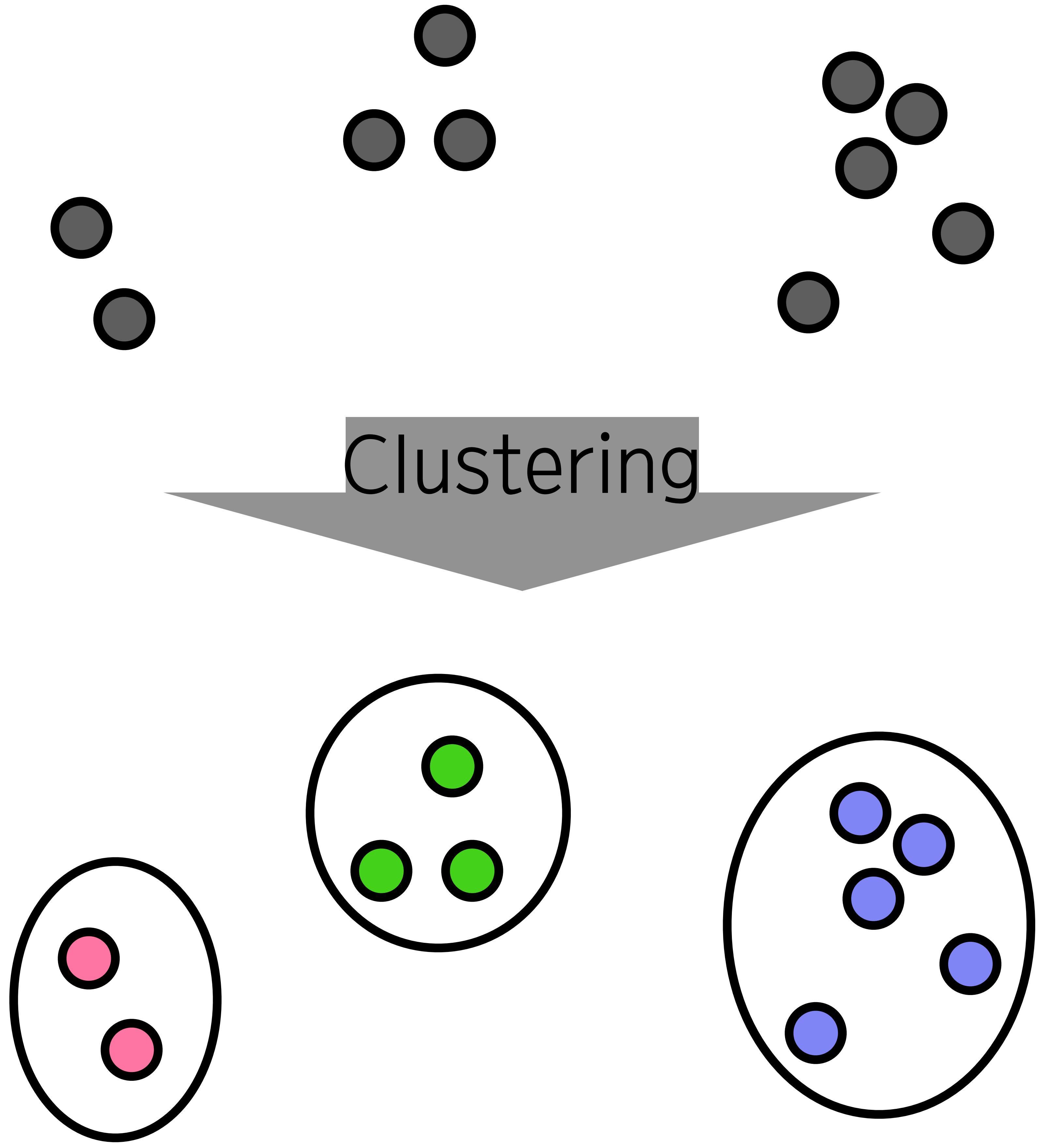
Supervised learning

Data with tag

	inputs			output
	Temperature	Wind (m/s)	Rain (mm)	Play tennis
Mon.	28	1	1	Yes
Tue.	23	2	5	No
Wed.	22	8	0	No
Thur.	21	3	0	Yes
	22	1	0	?

Unsupervised learning

Data without tag



Classification problem

Here is a tennis record of someone. Can you guess this person will play the tennis today?

	Temperature (℃)	Wind (m/s)	Rain (mm)	Play tennis
Mon.	28	1	1	Yes
Tue.	23	2	5	No
Wed.	22	8	0	No
Thur.	21	3	0	Yes
	22	1	0	?

Classification problem

Here is a tennis record of someone. Can you guess this person will play the tennis today?

	inputs			output
	Temperature (℃)	Wind (m/s)	Rain (mm)	Play tennis
Mon.	28	1	1	Yes
Tue.	23	2	5	No
Wed.	22	8	0	No
Thur.	21	3	0	Yes
	22	1	0	?

Classification problem

Here is a tennis record of someone. Can you guess this person will play the tennis today?

Training set

	Tempera ture (℃)	Wind (m/s)	Rain (mm)	Play tennis
Mon.	28	1	1	Yes
Tue.	23	2	5	No
Wed.	22	8	0	No
Thur.	21	3	0	Yes
	22	1	0	?

Data set: $D_i = (x_i, y_i)$

$x_1 = (28,1,1),\ y_1 = 1$

$x_2 = (23,2,5),\ y_2 = 0$

$x_3 = (22,8,0),\ y_3 = 0$

$x_4 = (21,3,0),\ y_4 = 1$

Classification problem

Here is a tennis record of someone. Can you guess this person will play the tennis today?

Training set

	Temperature (℃)	Wind (m/s)	Rain (mm)	Play tennis
Mon.	28	1	1	Yes
Tue.	23	2	5	No
Wed.	22	8	0	No
Thur.	21	3	0	Yes
	22	1	0	?

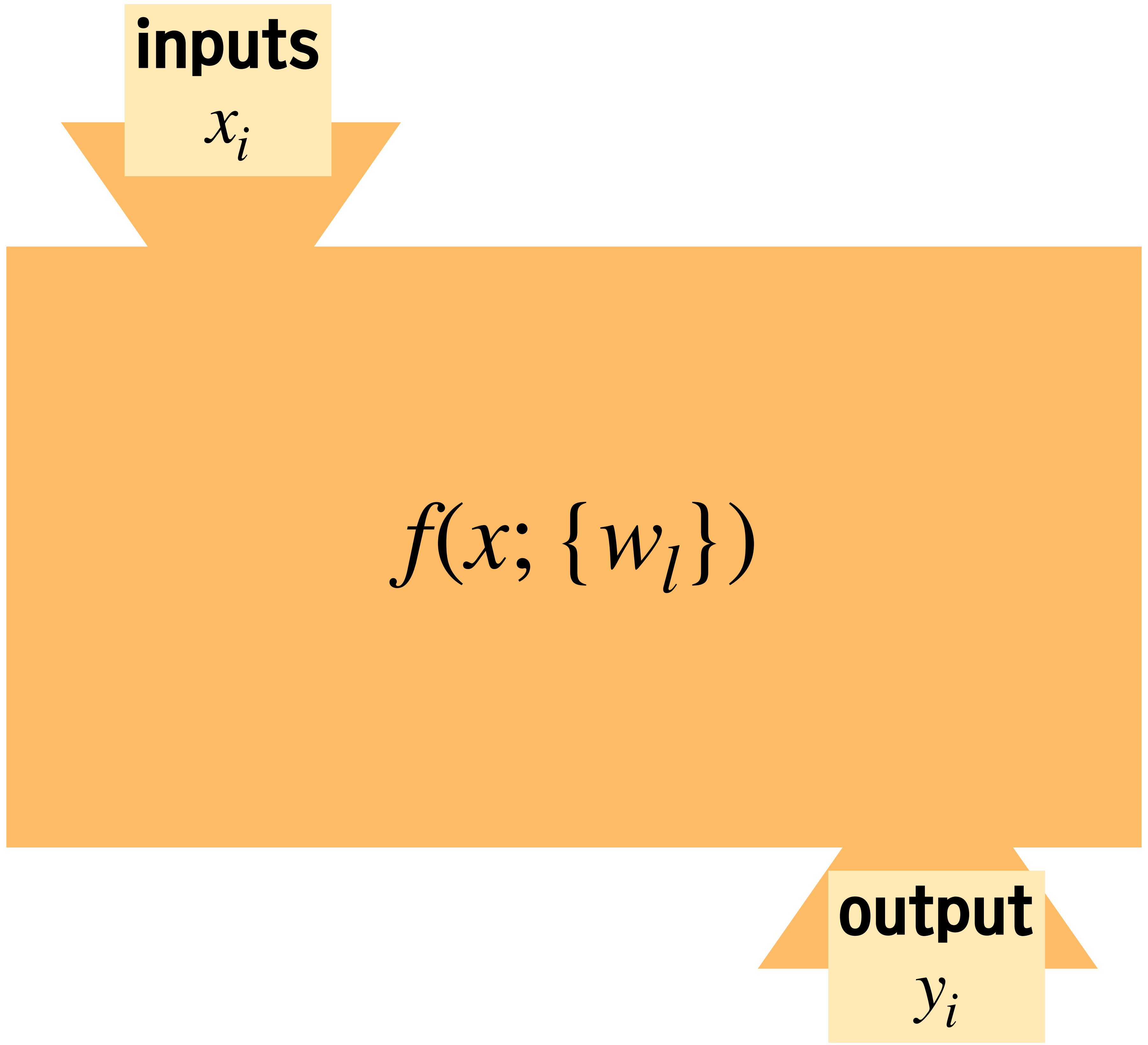
Data set: $D_i = (x_i, y_i)$

$x_1 = (28,1,1),\ y_1 = 1$

$x_2 = (23,2,5),\ y_2 = 0$

$x_3 = (22,8,0),\ y_3 = 0$

$x_4 = (21,3,0),\ y_4 = 1$



Classification problem

Here is a tennis record of someone. Can you guess this person will play the tennis today?

Training set

	Temperature (℃)	Wind (m/s)	Rain (mm)	Play tennis
Mon.	28	1	1	Yes
Tue.	23	2	5	No
Wed.	22	8	0	No
Thur.	21	3	0	Yes
	22	1	0	?

Data set: $D_i = (x_i, y_i)$

$x_1 = (28,1,1),\ y_1 = 1$

$x_2 = (23,2,5),\ y_2 = 0$

$x_3 = (22,8,0),\ y_3 = 0$

$x_4 = (21,3,0),\ y_4 = 1$

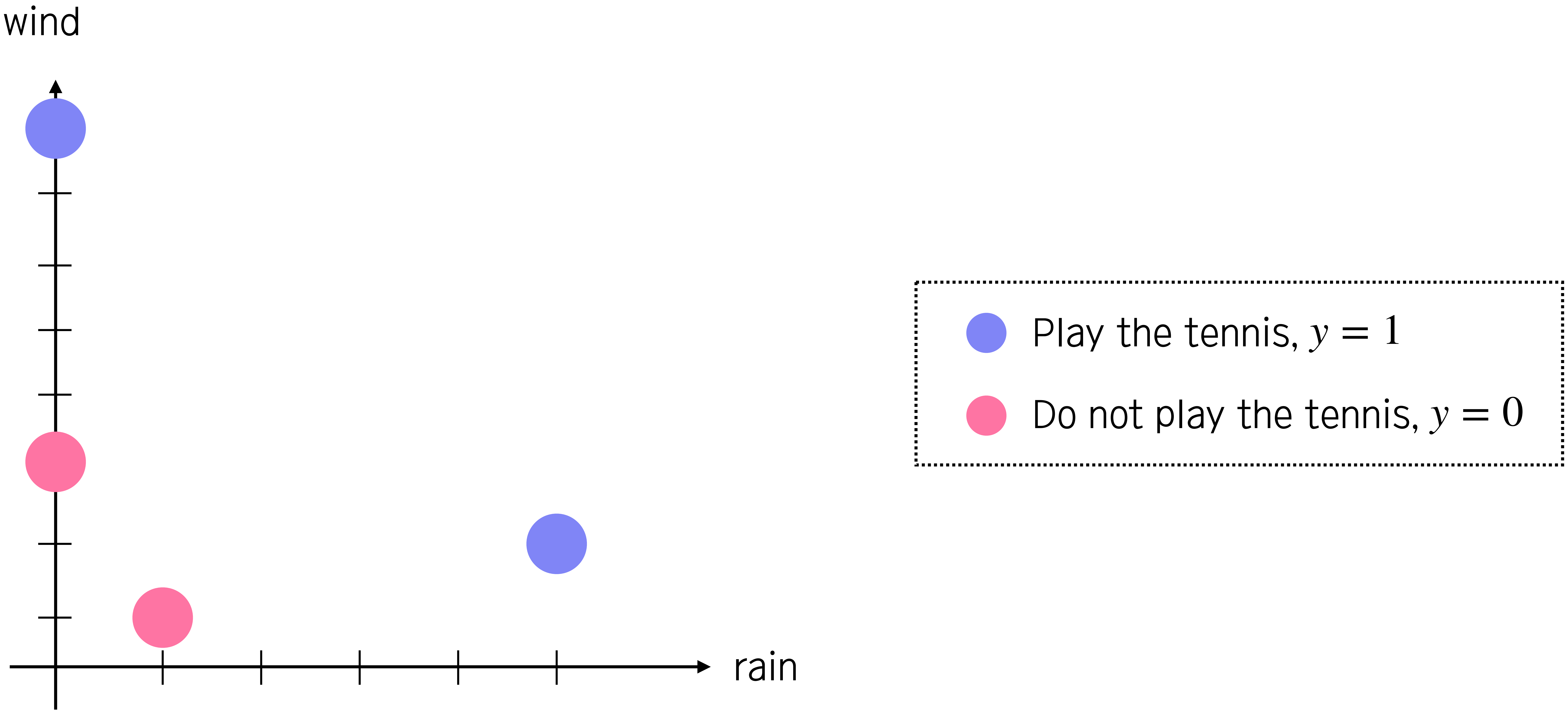
inputs
x_i

Using training set, obtain $\{w_l\}$

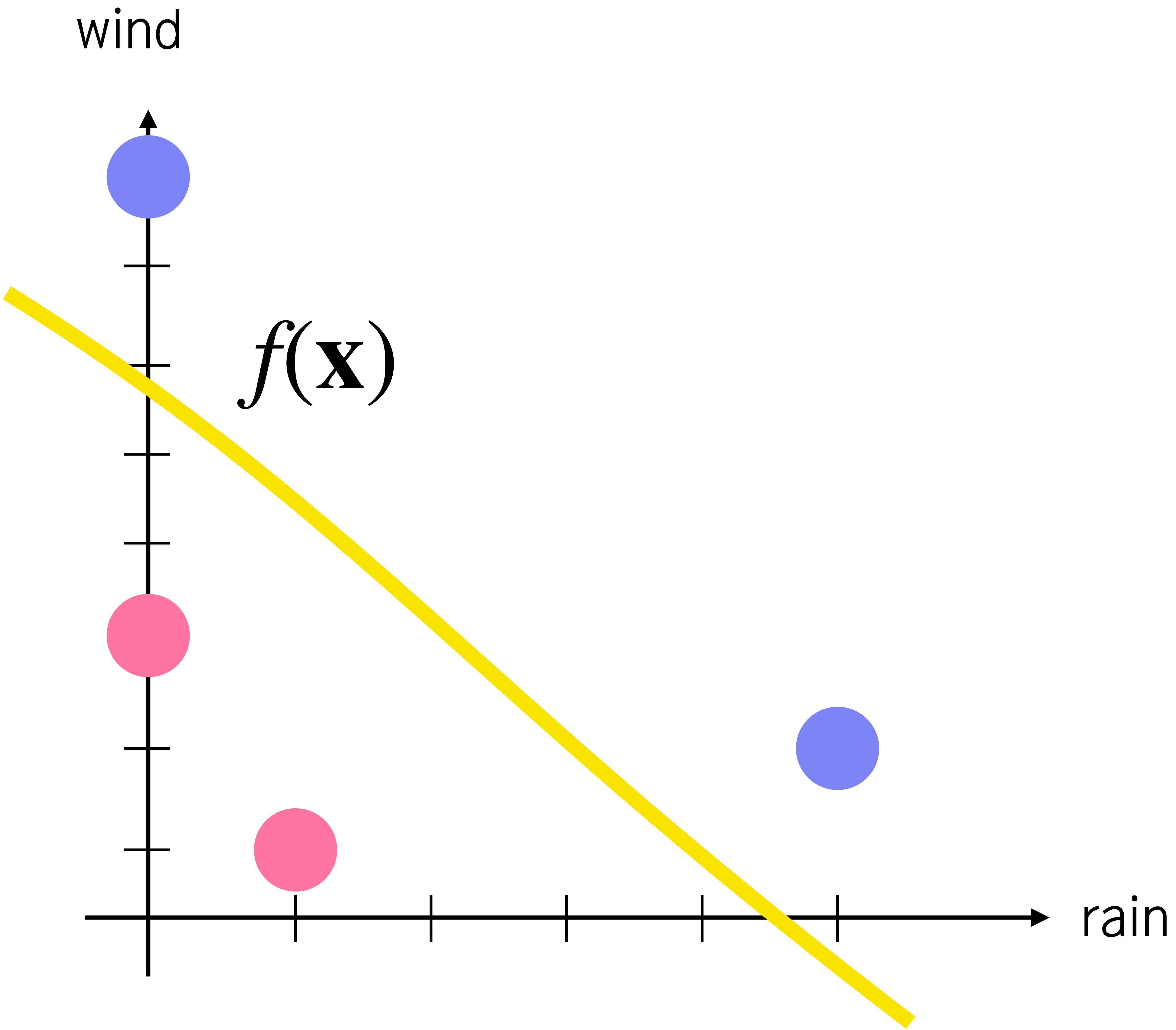
$\boxed{f(x; \{w_l\})}$

output
y_i

Classification problem



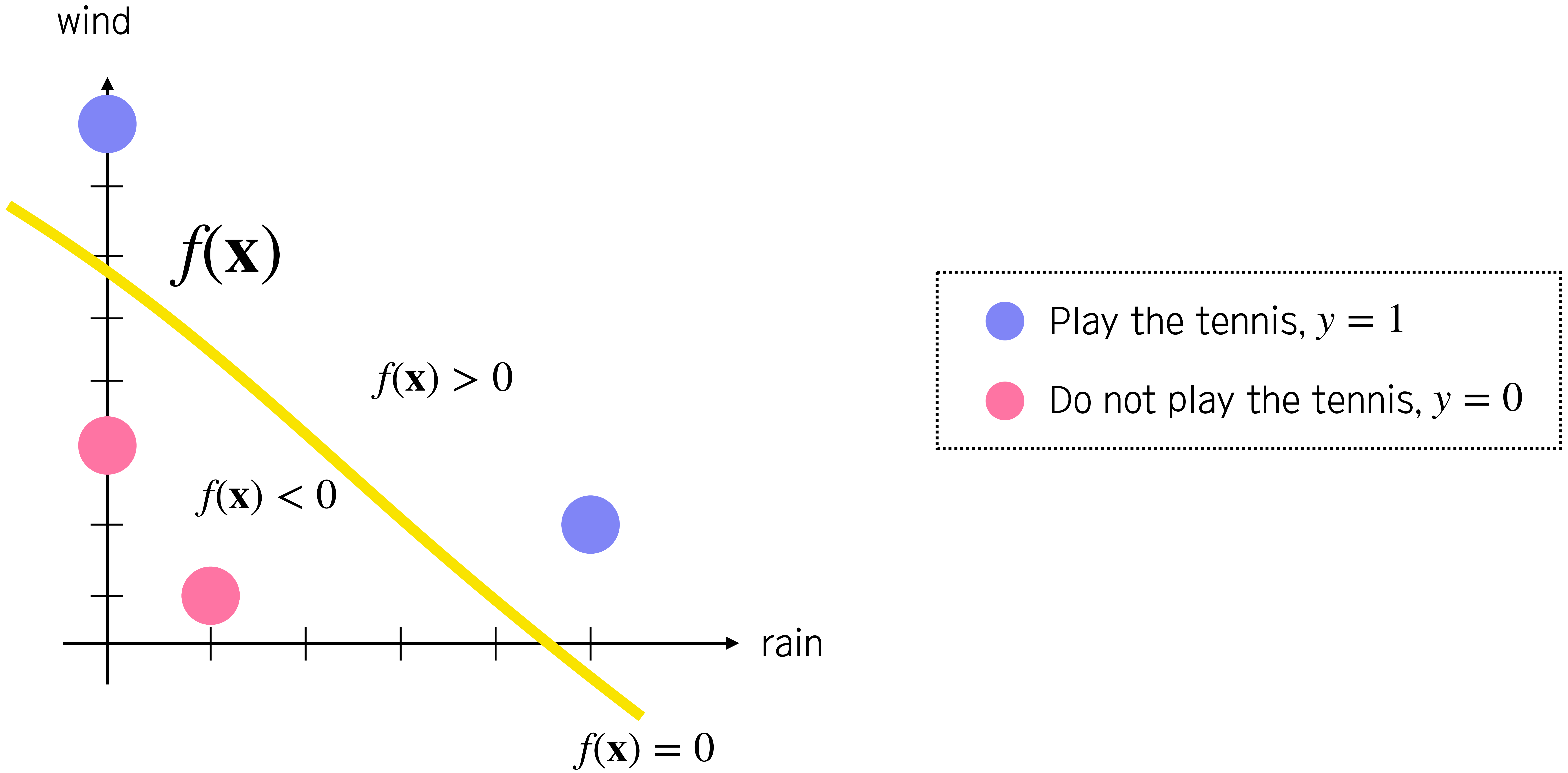
Classification problem



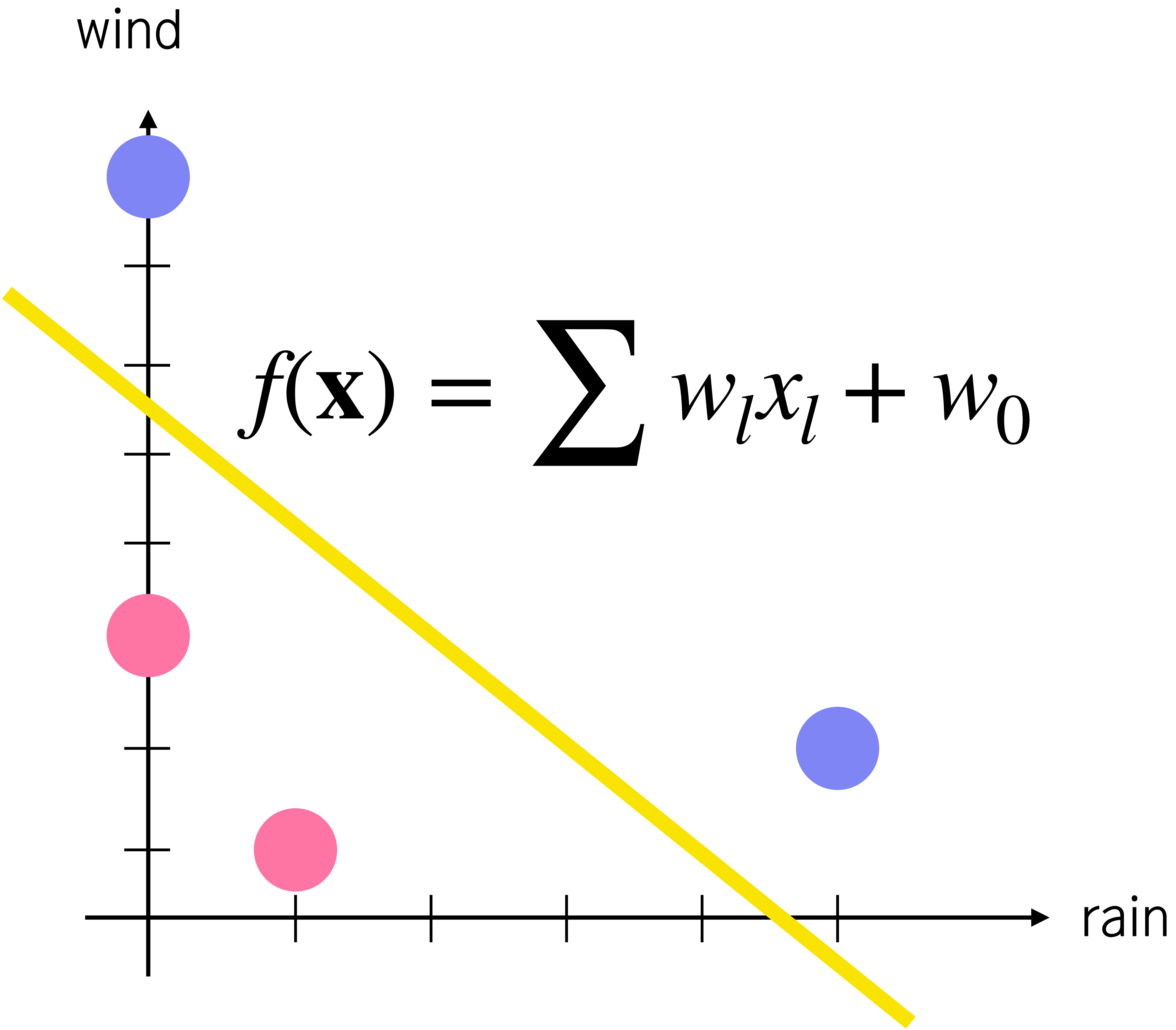
Play the tennis, $y = 1$

Do not play the tennis, $y = 0$

Classification problem

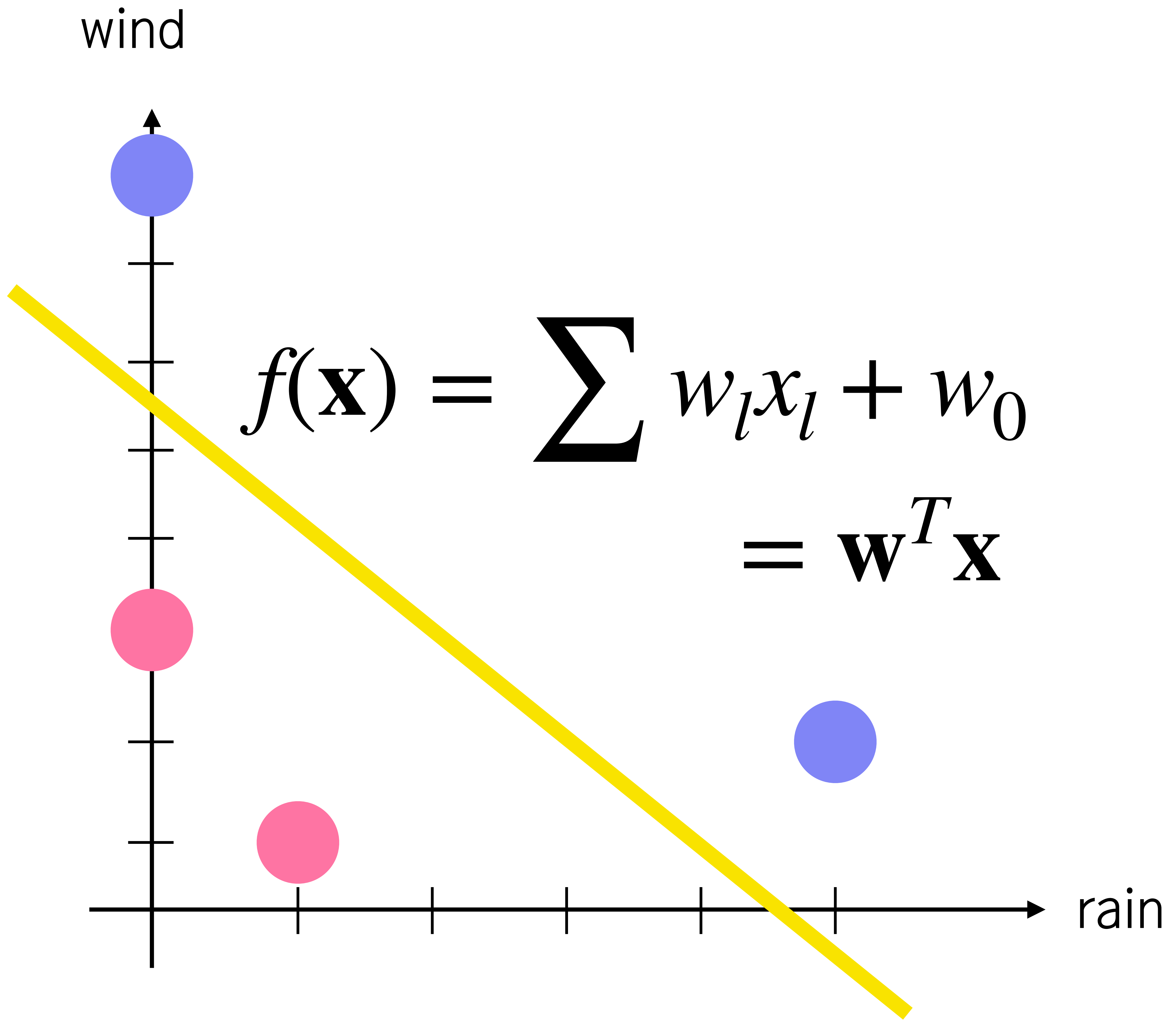


Linear Classification



$$f(\mathbf{x}) = \sum w_l x_l + w_0$$

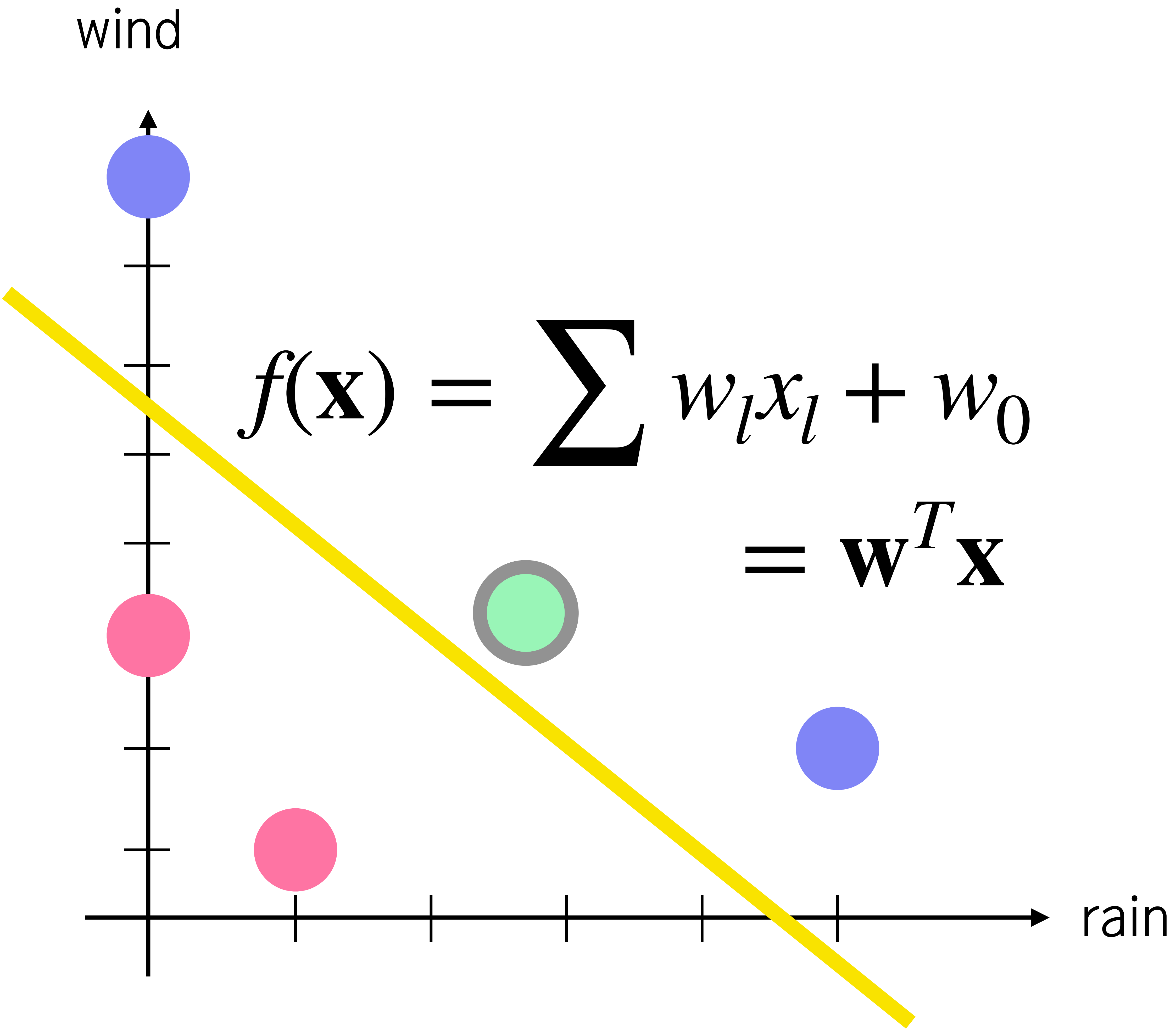
Linear Classification



$$f(\mathbf{x}) = \sum w_l x_l + w_0$$

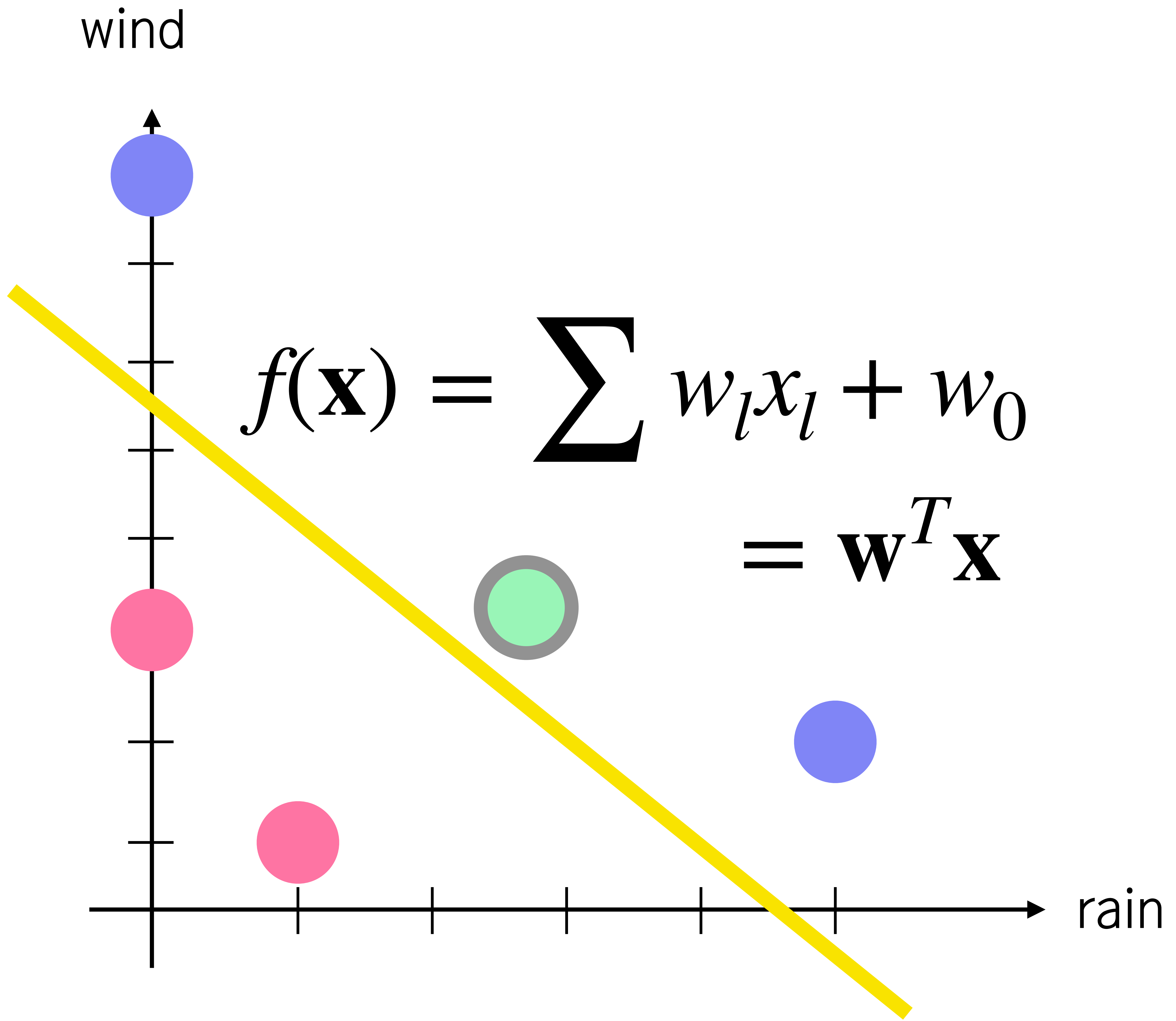
$$= \mathbf{w}^T \mathbf{x}$$

Linear Classification



$$f(\mathbf{x}) = \sum w_l x_l + w_0$$
$$= \mathbf{w}^T \mathbf{x}$$

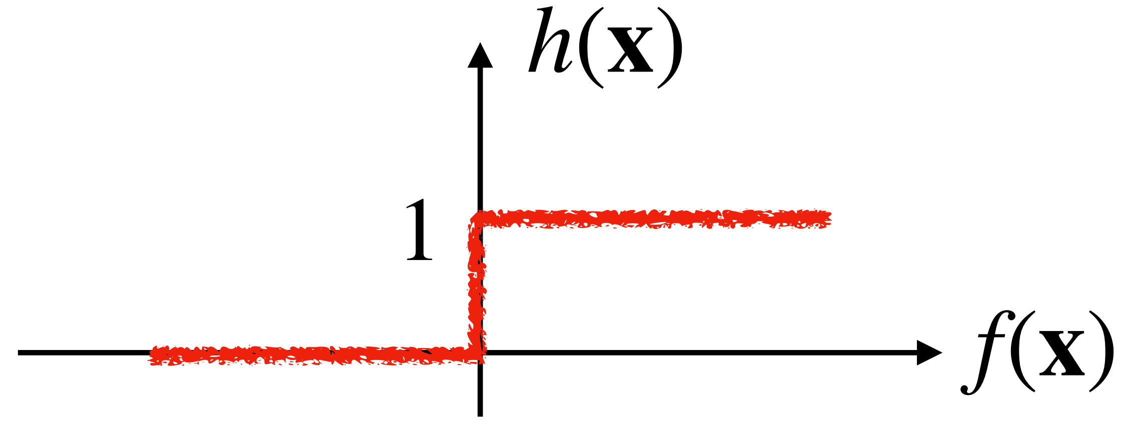
Linear Classification



$$f(\mathbf{x}) = \sum w_l x_l + w_0$$
$$= \mathbf{w}^T \mathbf{x}$$

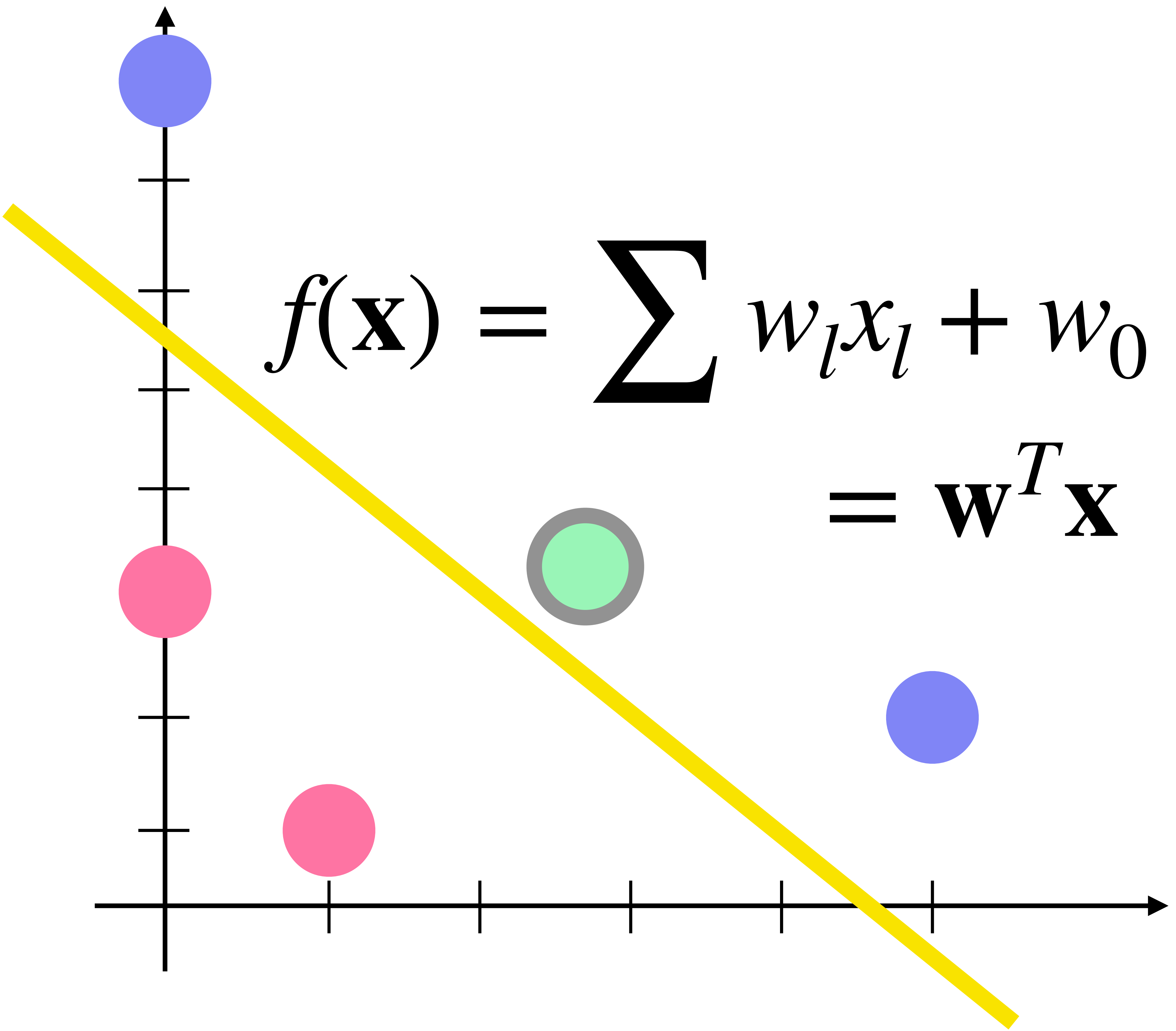
Activation function

$$h(\mathbf{x}) = \begin{cases} 1 & \text{for } \mathbf{w}^T\mathbf{x} > 0 \\ 0 & \text{otherwise} \end{cases}$$



Activation function decides the outcome based on $\mathbf{w}^T\mathbf{x}$.

Linear Classification

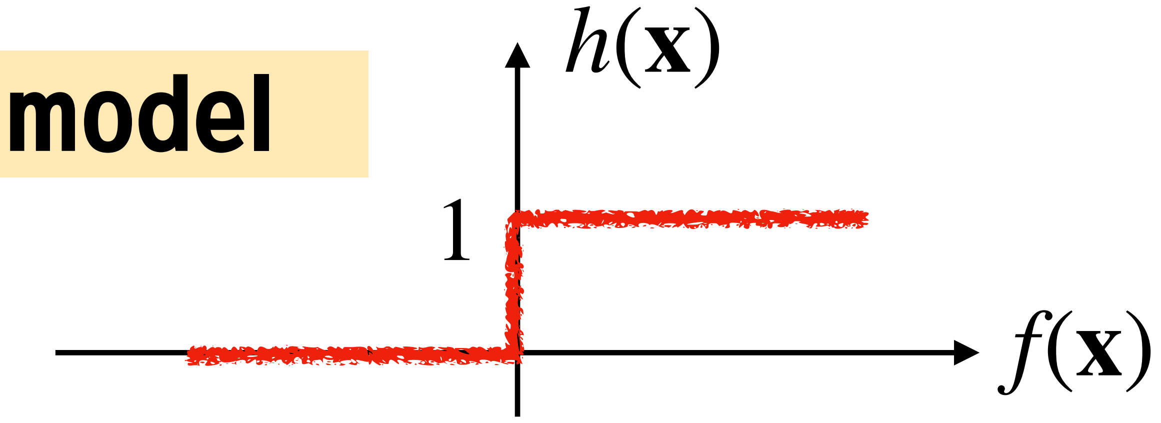


$$f(\mathbf{x}) = \sum w_l x_l + w_0$$
$$= \mathbf{w}^T\mathbf{x}$$

Activation function

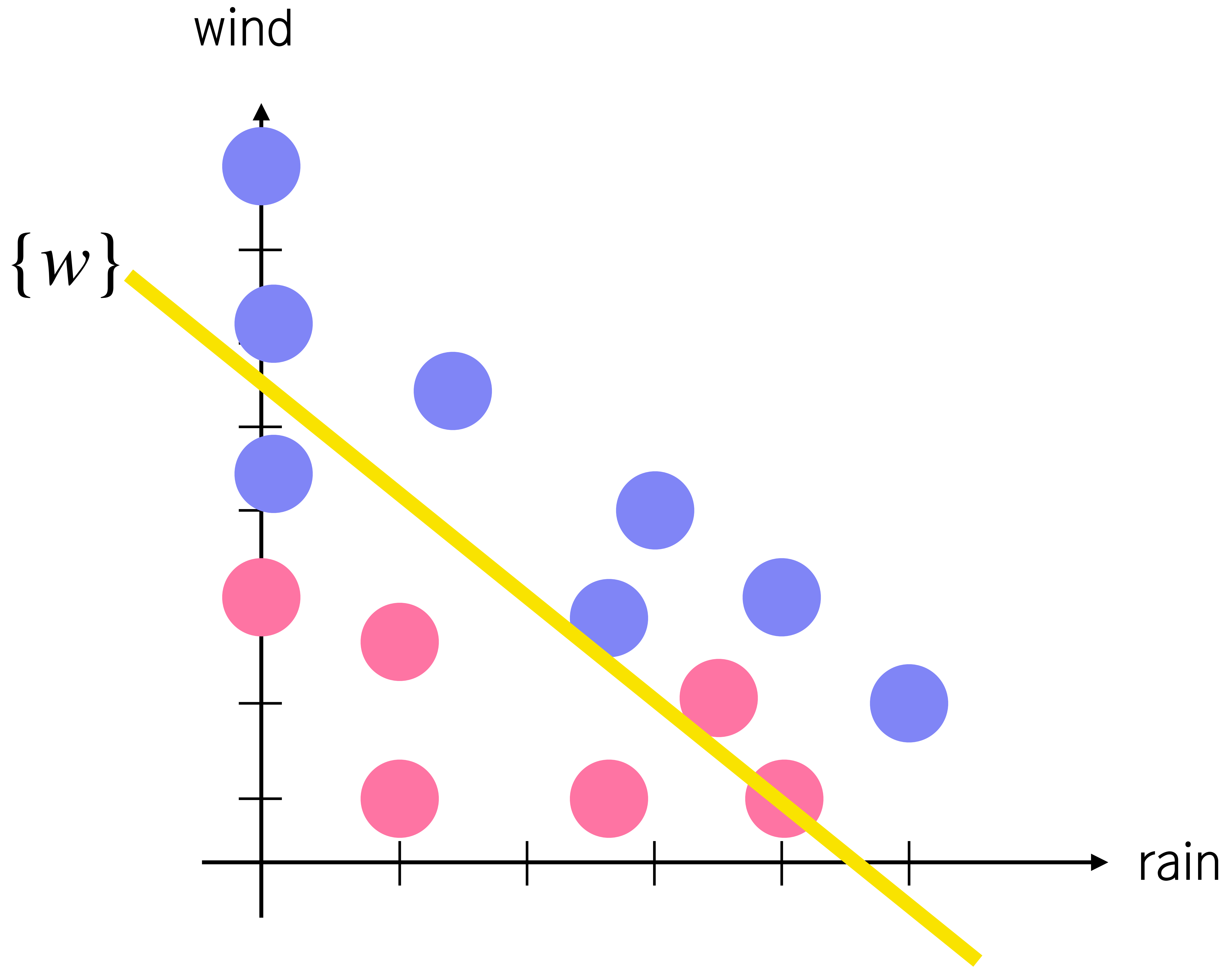
$$h(\mathbf{x}) = \begin{cases} 1 & \text{for } \mathbf{w}^T\mathbf{x} > 0 \\ 0 & \text{otherwise} \end{cases}$$

McCluch-Pitts model

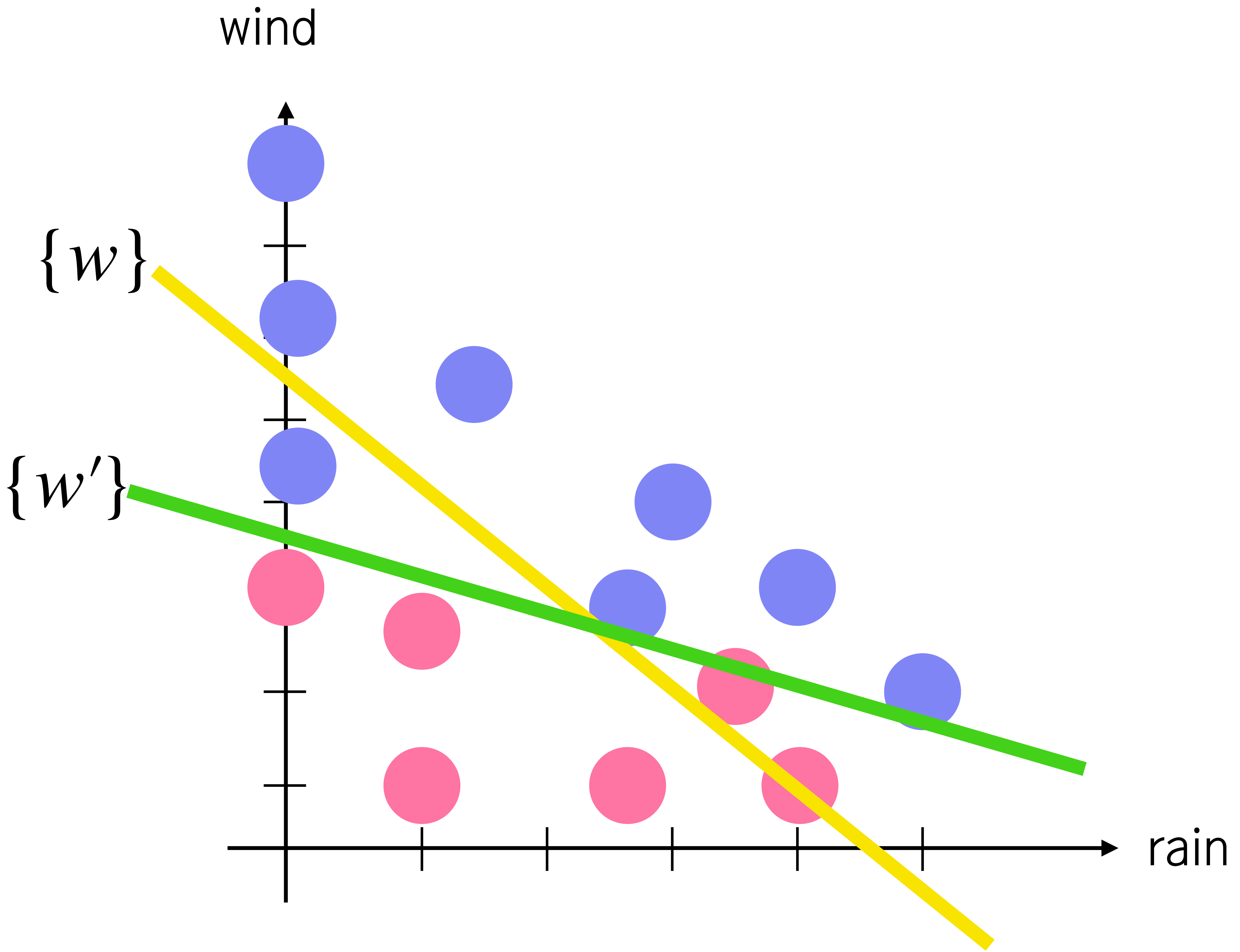


Activation function decides the outcome based on $\mathbf{w}^T\mathbf{x}$.

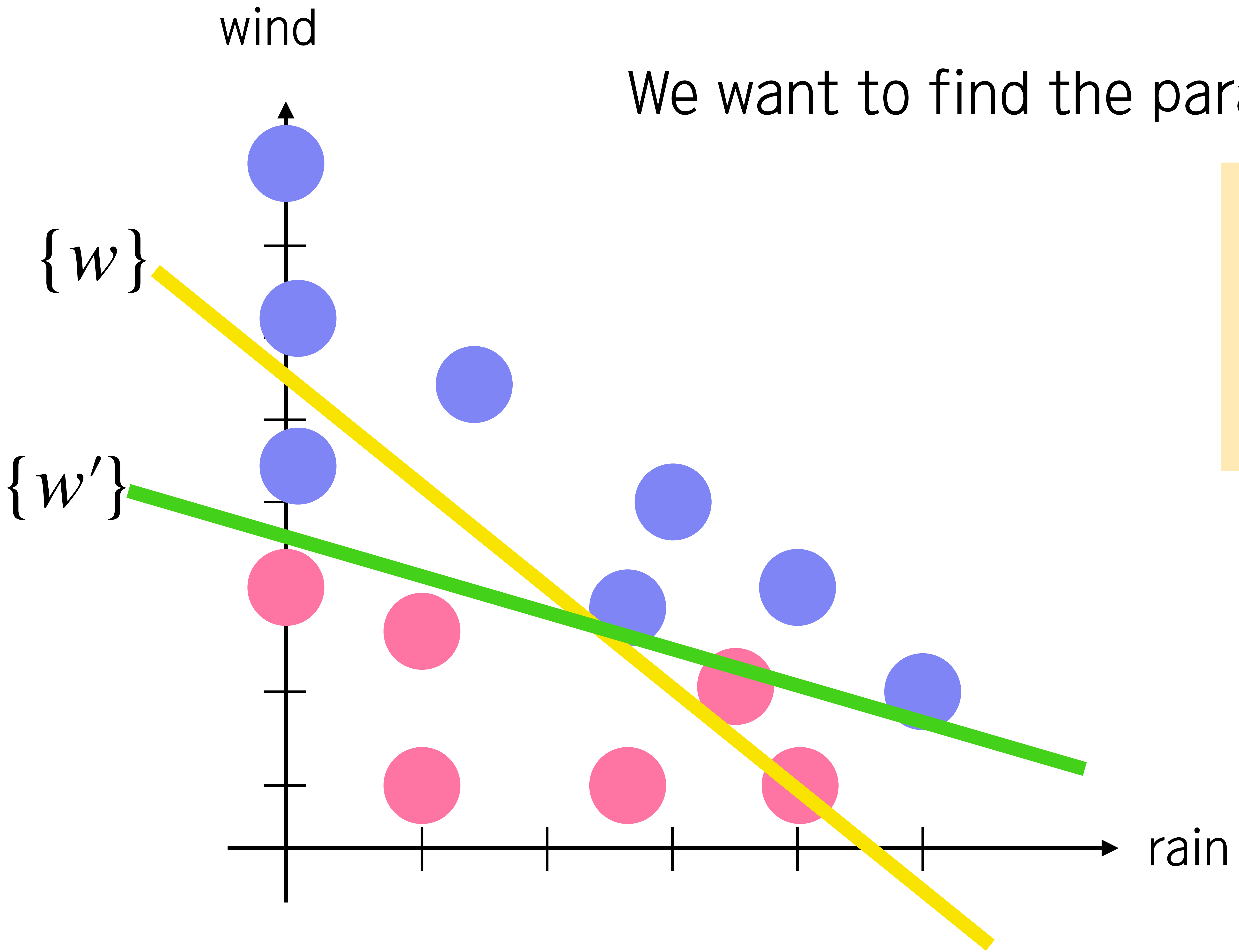
Linear Classification



Linear Classification



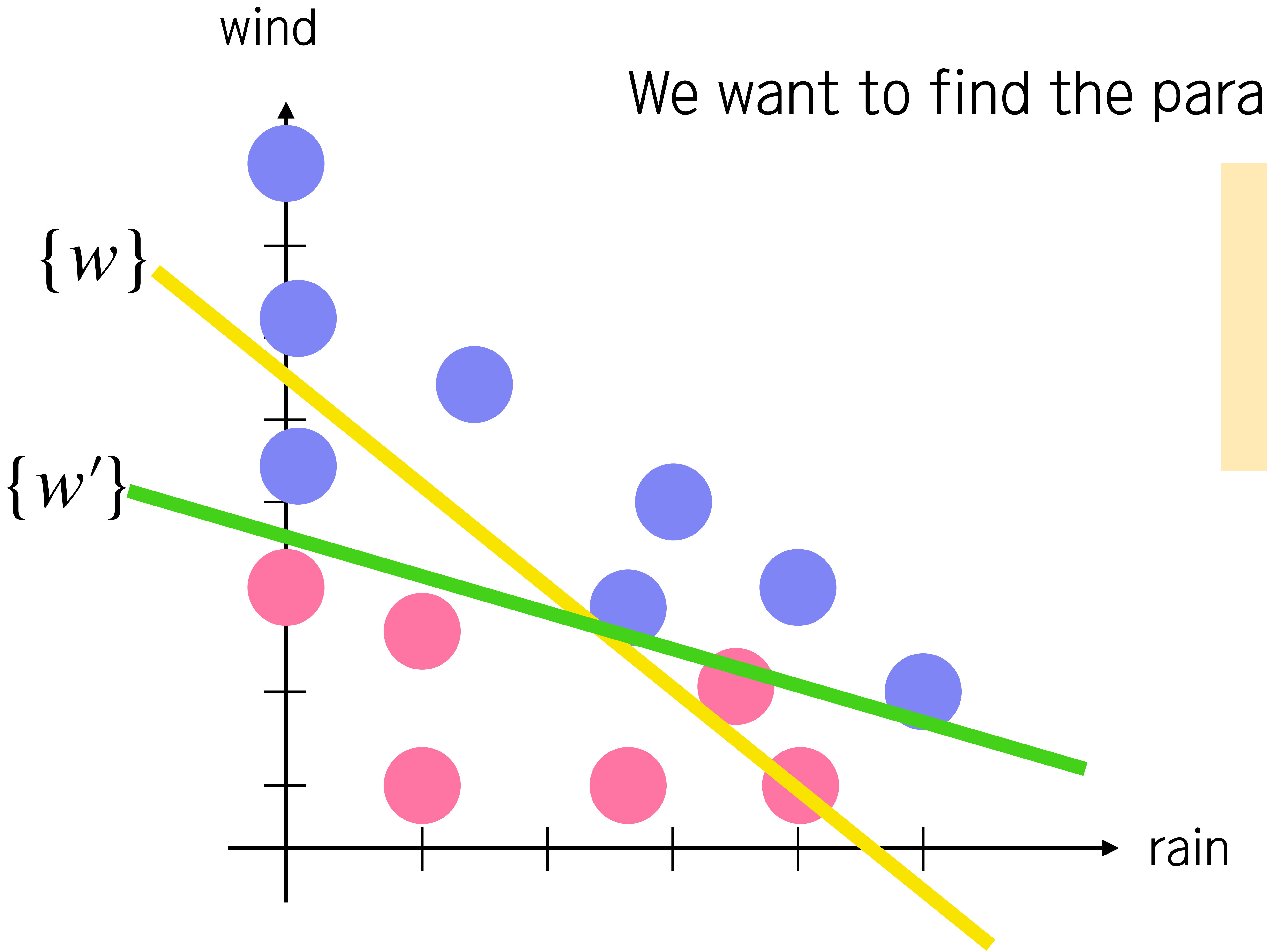
Linear Classification



We want to find the parameter set $\{w^*\}$ to minimize the error.

Difference	$h(\mathbf{x}_i) - y_i$
Error	$E = \sum_i \left(h(\mathbf{x}_i) - y_i\right)^2$

Linear Classification



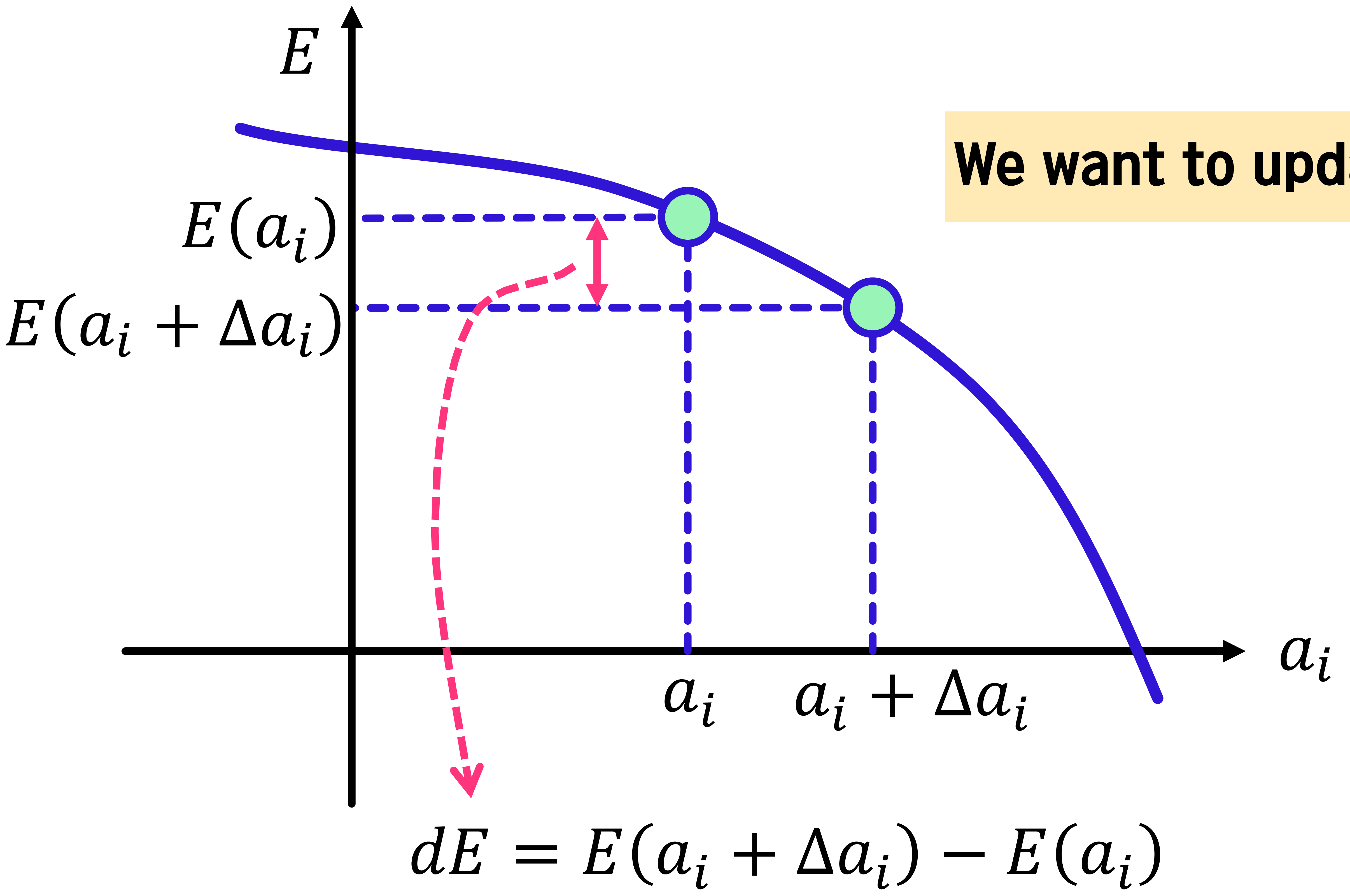
We want to find the parameter set $\{w^*\}$ to minimize the error.

Difference	$h(\mathbf{x}_i) - y_i$
Error	$E = \sum_i \left(h(\mathbf{x}_i) - y_i\right)^2$

⇨ Gradient descent method!

Gradient descent method (경사하강법)

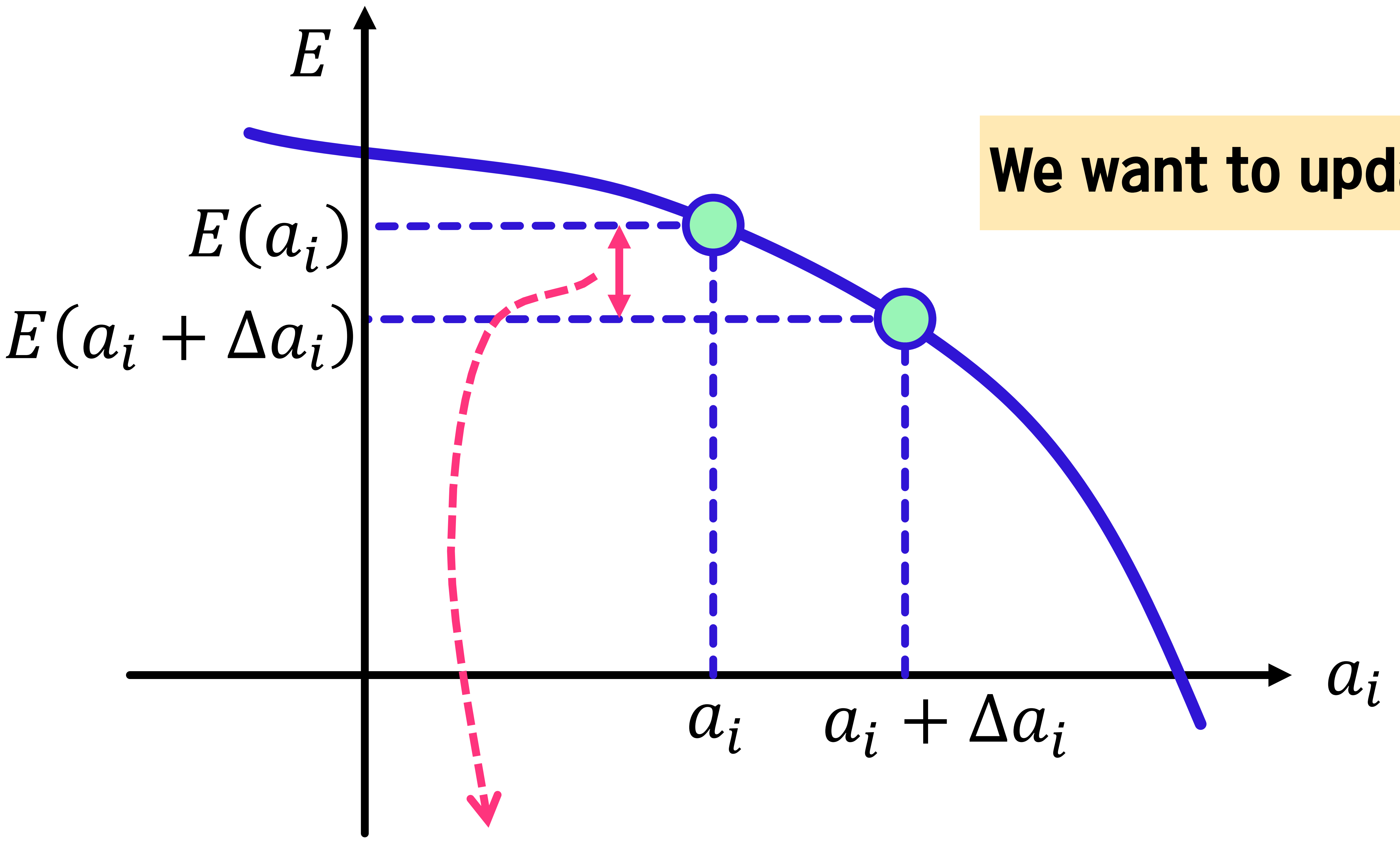
Starting from an initial parameter set $\{a_i\}$, we can update $a_i' = a_i + \Delta a_i$ to reduce the error E.



We want to update a_i such that $dE < 0$.

Gradient descent method (경사하강법)

Starting from an initial parameter set $\{a_i\}$, we can update $a_i' = a_i + \Delta a_i$ to reduce the error E.



$$dE = E(a_i + \Delta a_i) - E(a_i)$$

We want to update a_i such that $dE < 0$.

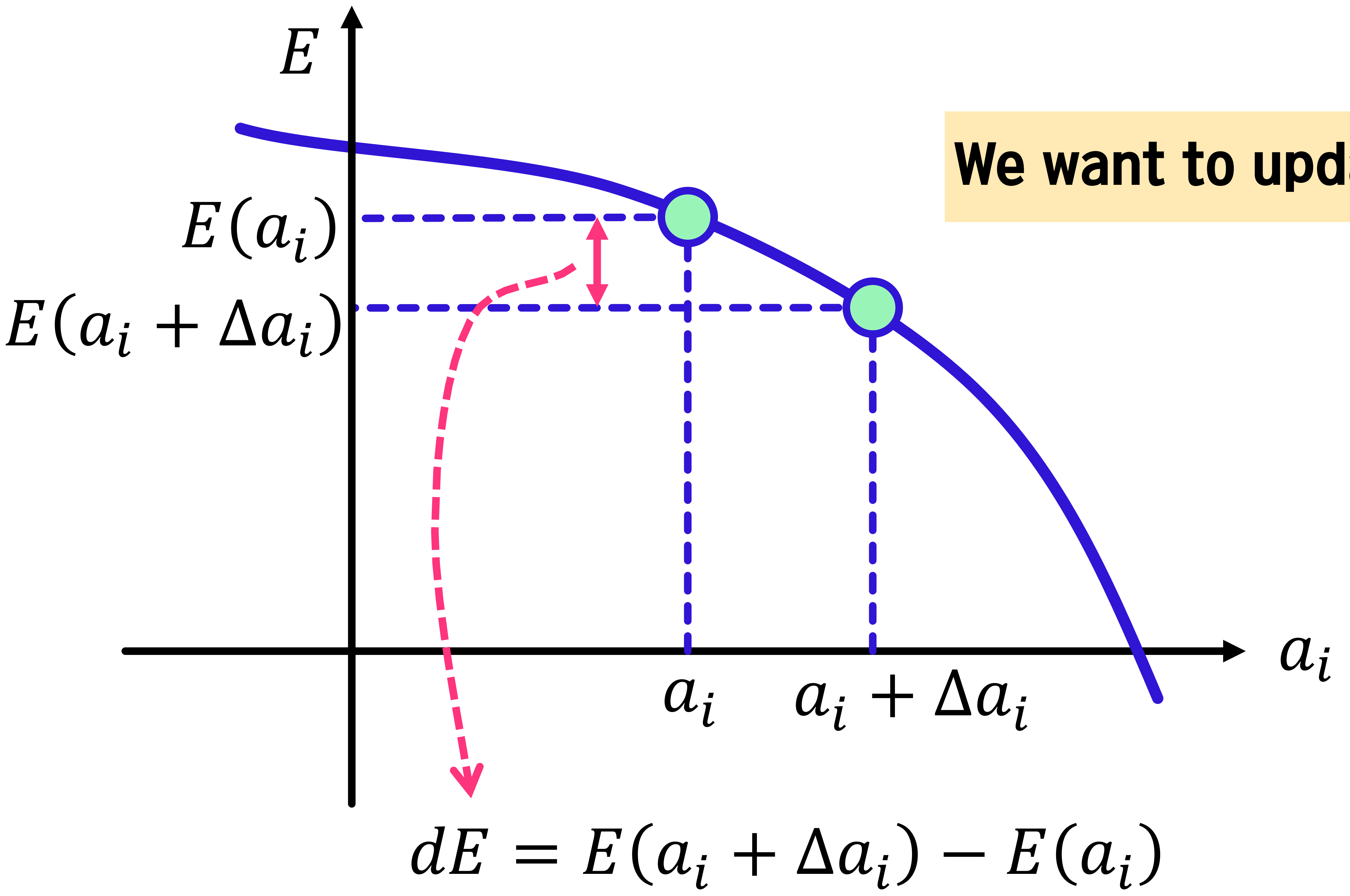
$$dE = E(\vec{a} + \Delta\vec{a}) - E(\vec{a}) = \nabla E \cdot \Delta\vec{a}$$

So if we update $\vec{a}$ to a direction of $(-\nabla E)$,

the error will be reduced.

Gradient descent method (경사하강법)

Starting from an initial parameter set $\{a_i\}$, we can update $a_i' = a_i + \Delta a_i$ to reduce the error E.



We want to update a_i such that $dE < 0$.

$$dE = E(\vec{a} + \Delta\vec{a}) - E(\vec{a}) = \nabla E \cdot \Delta\vec{a}$$

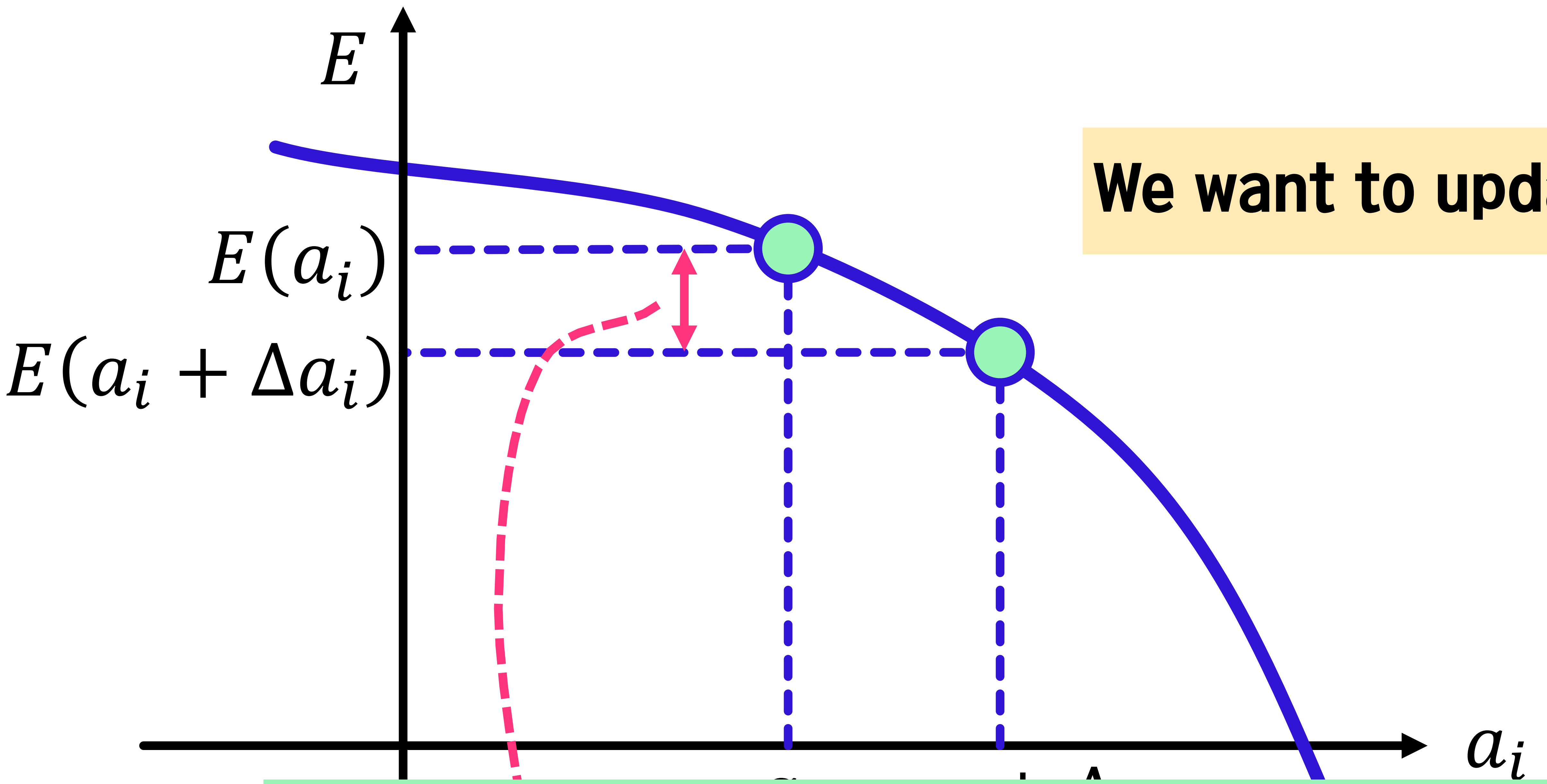
So if we update $\vec{a}$ to a direction of $(-\nabla E)$,

the error will be reduced.

$$a_i \rightarrow a_i - c\frac{\partial E}{\partial a_i}$$

Gradient descent method (경사하강법)

Starting from an initial parameter set $\{a_i\}$, we can update $a_i' = a_i + \Delta a_i$ to reduce the error E.



We want to update a_i such that $dE < 0$.

$$dE = E(\vec{a} + \Delta\vec{a}) - E(\vec{a}) = \nabla E \cdot \Delta\vec{a}$$

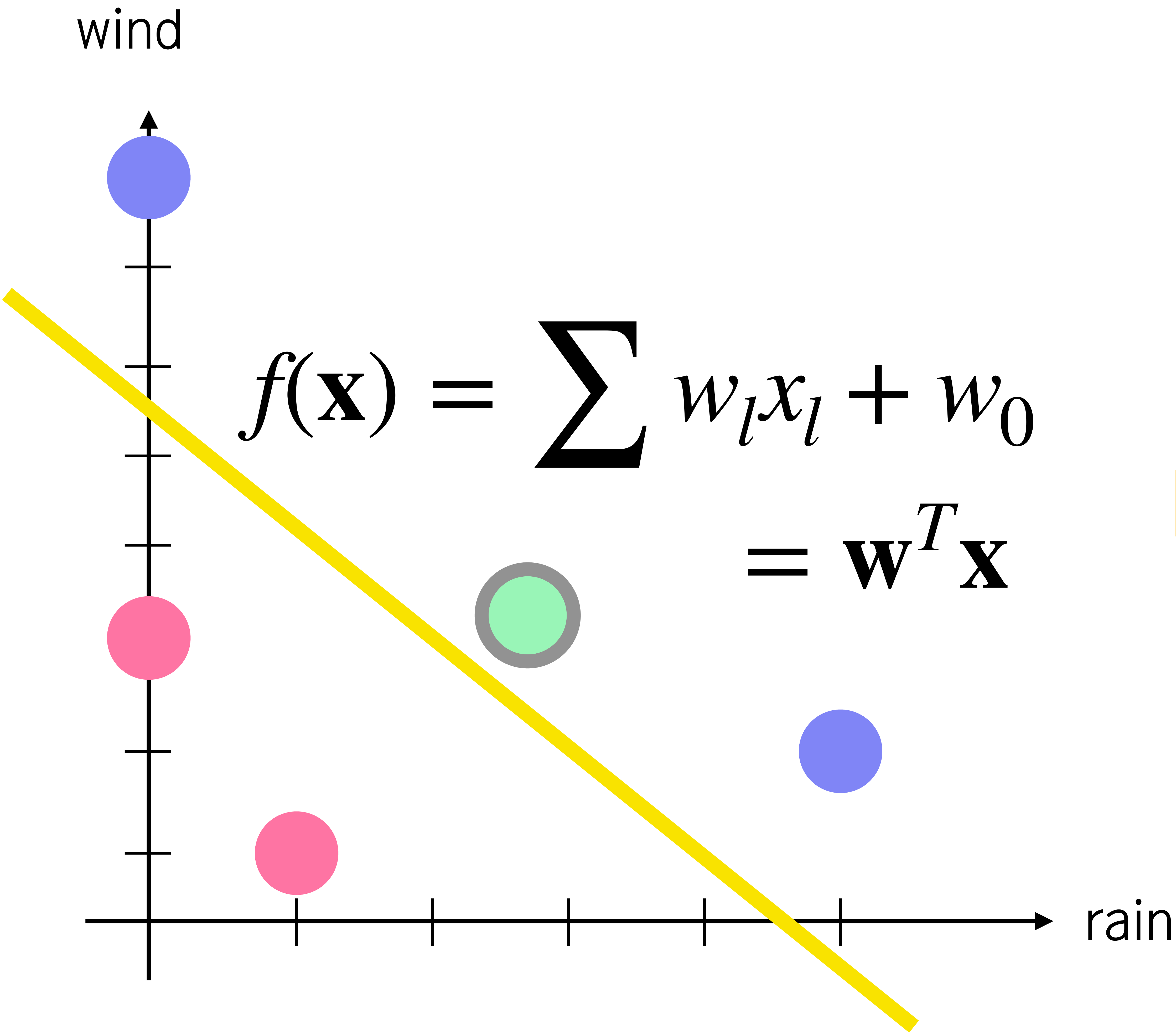
So if we update $\vec{a}$ to a direction of $(-\nabla E)$,

the error will be reduced.

$$a_i \to a_i - c\frac{\partial E}{\partial a_i}$$

Machine learns themselves what parameters should they have to get the small error according to the given algorithm. So, we call it learning and call c learning rate.

Linear Classification

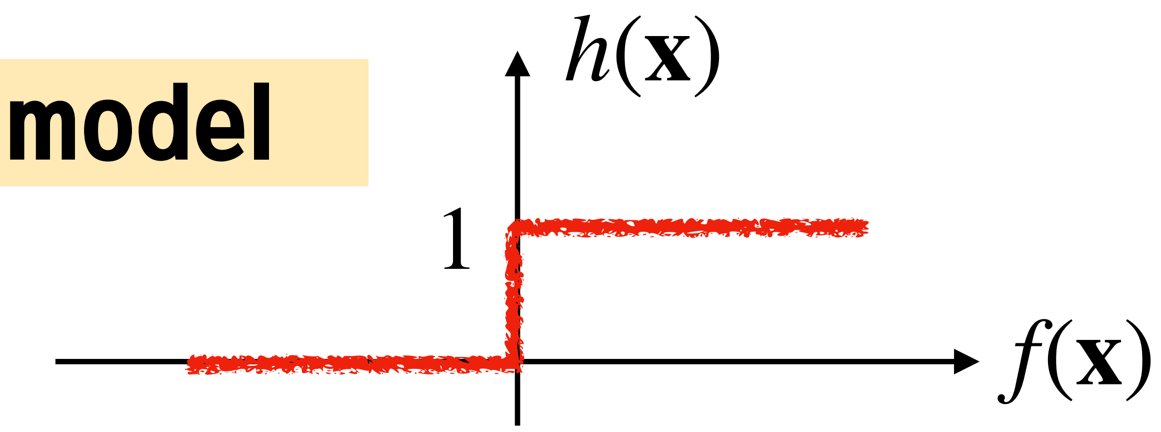


$$f(\mathbf{x}) = \sum w_l x_l + w_0$$
$$= \mathbf{w}^T\mathbf{x}$$

Activation function

$$h(\mathbf{x}) = \begin{cases} 1 & \text{for } \mathbf{w}^T\mathbf{x} > 0 \\ 0 & \text{otherwise} \end{cases}$$

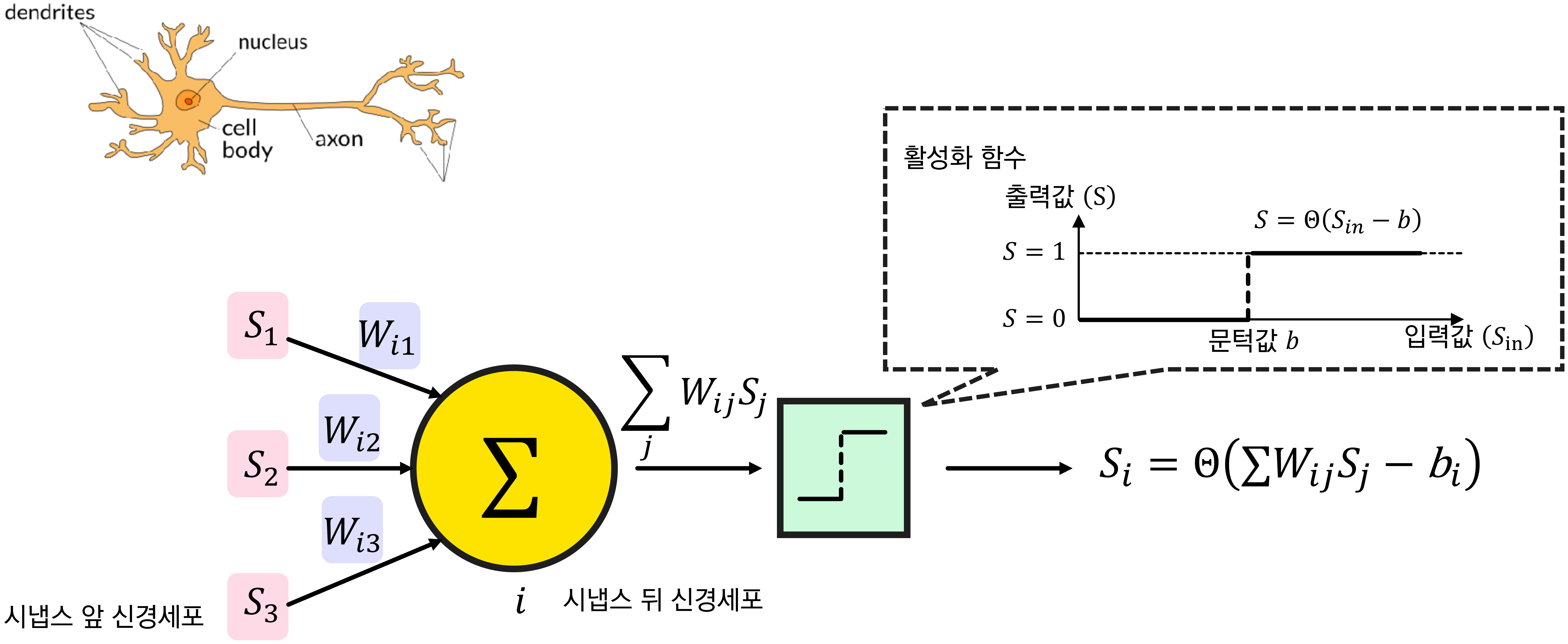
McCluch-Pitts model



Activation function decides the outcome based on $\mathbf{w}^T\mathbf{x}$.

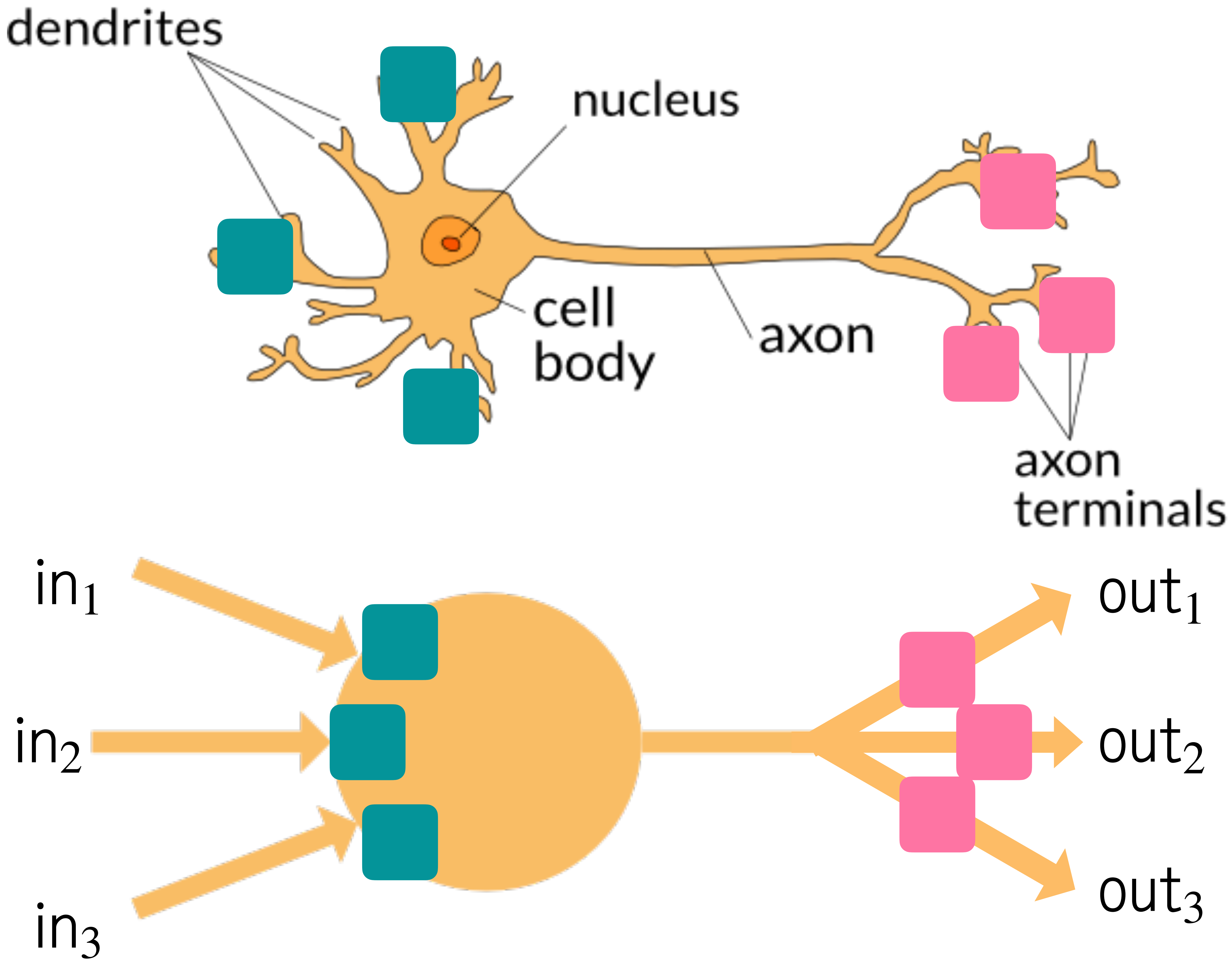
Schematic figure for McCluch-Pitts model

A model that mimics a nerve cell that collects input information from multiple nerve cells and determines whether or not to fire.



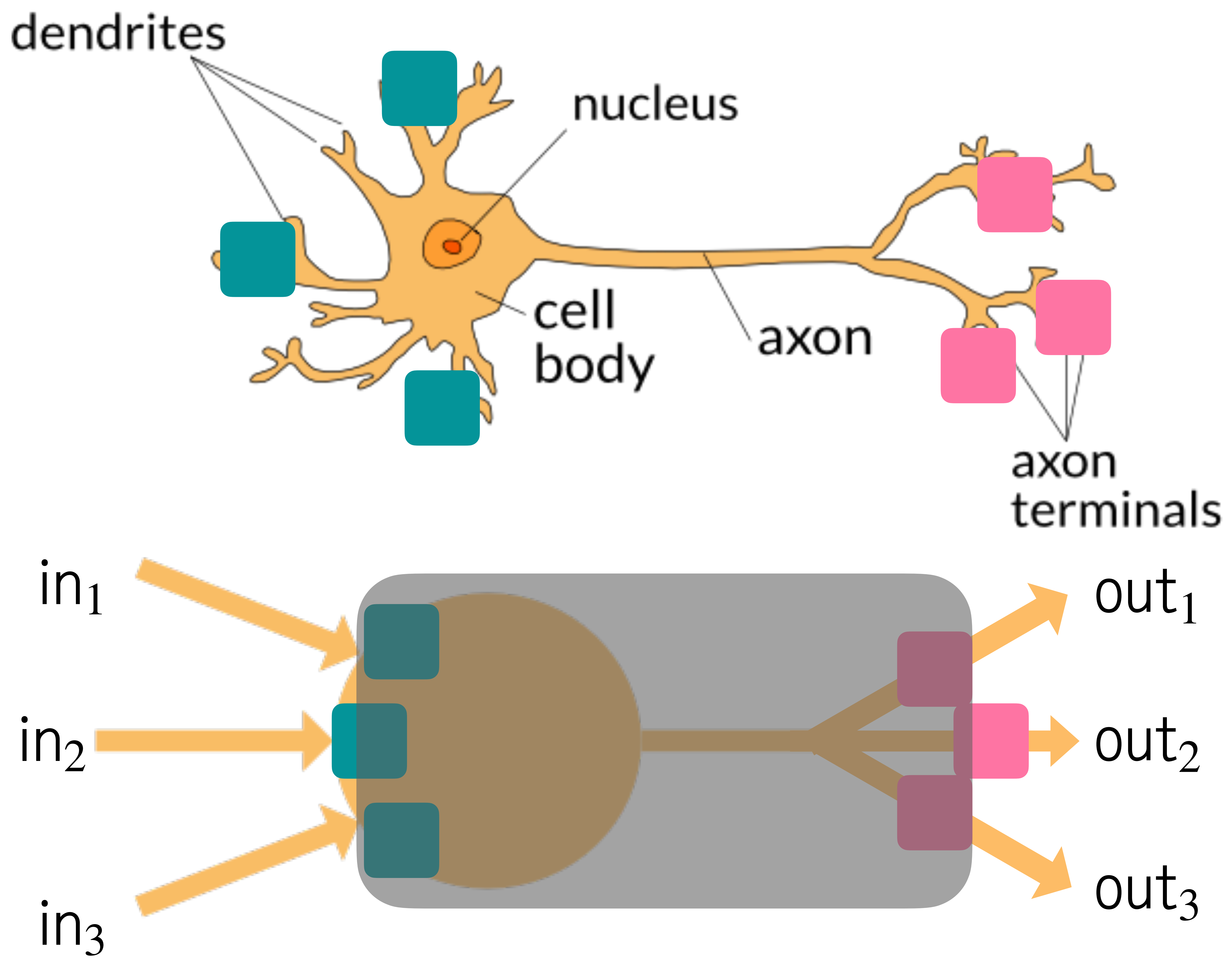
Perceptron

We can imagine two parts of the perceptron: input nodes and output nodes



Perceptron

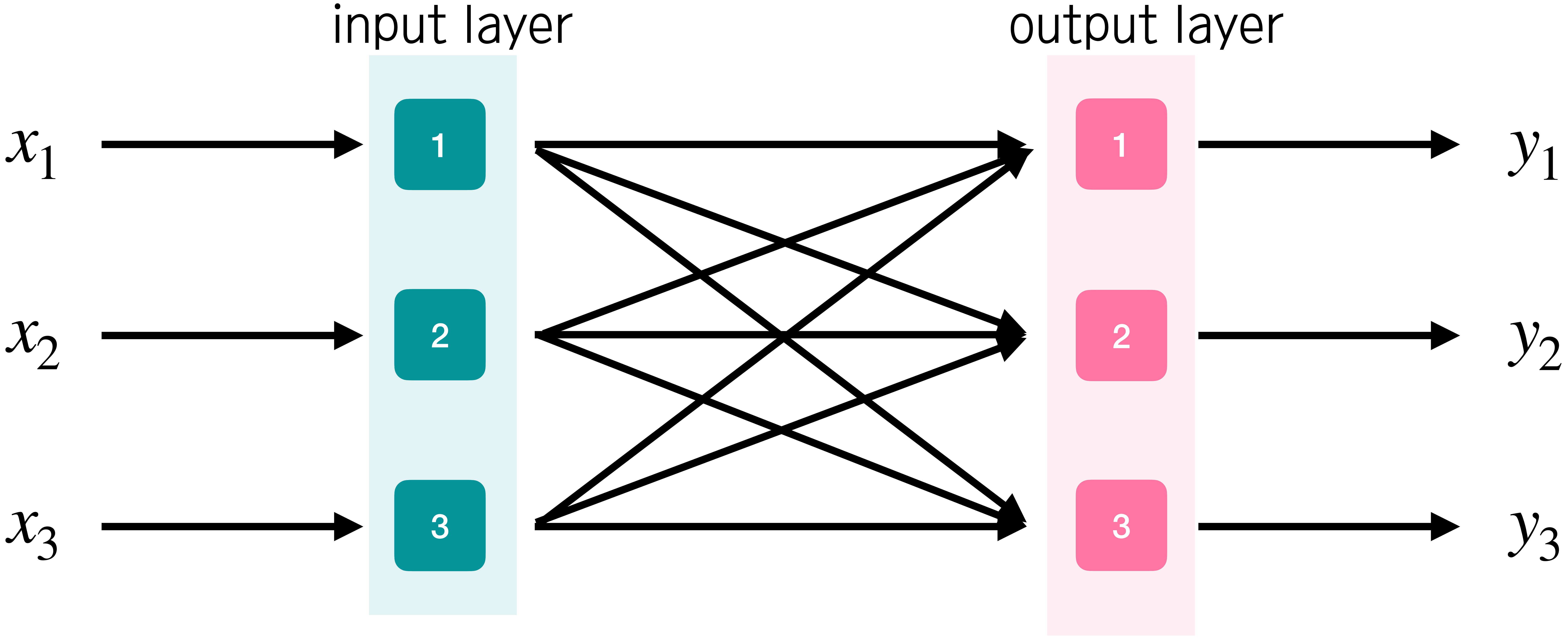
We can imagine two parts of the perceptron: input nodes and output nodes
Perceptron algorithm is just a mapping from input signals to output signals



Perceptron

Two-layer system: input and output layers

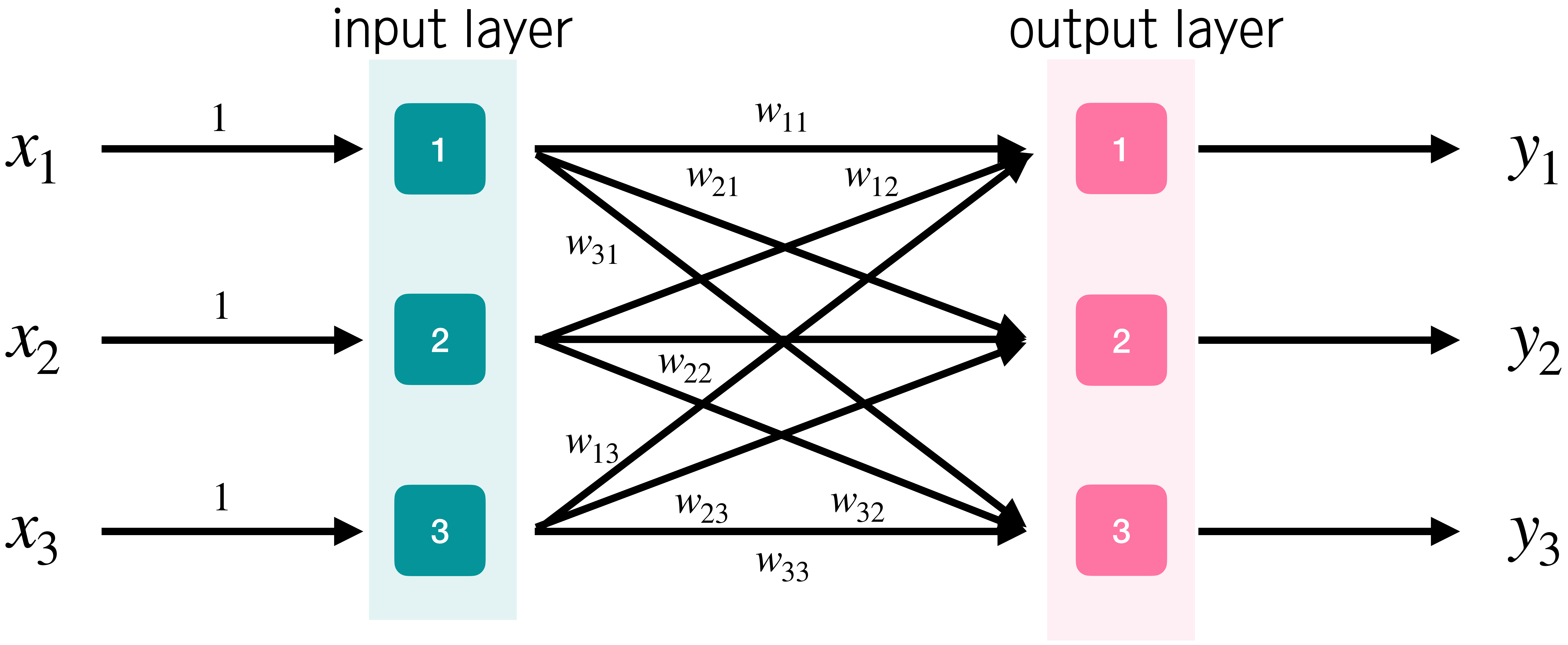
We can choose the number of nodes in each layer (free parameters)



Perceptron

Two-layer system: input and output layers

We can choose the number of nodes in each layer (free parameters)

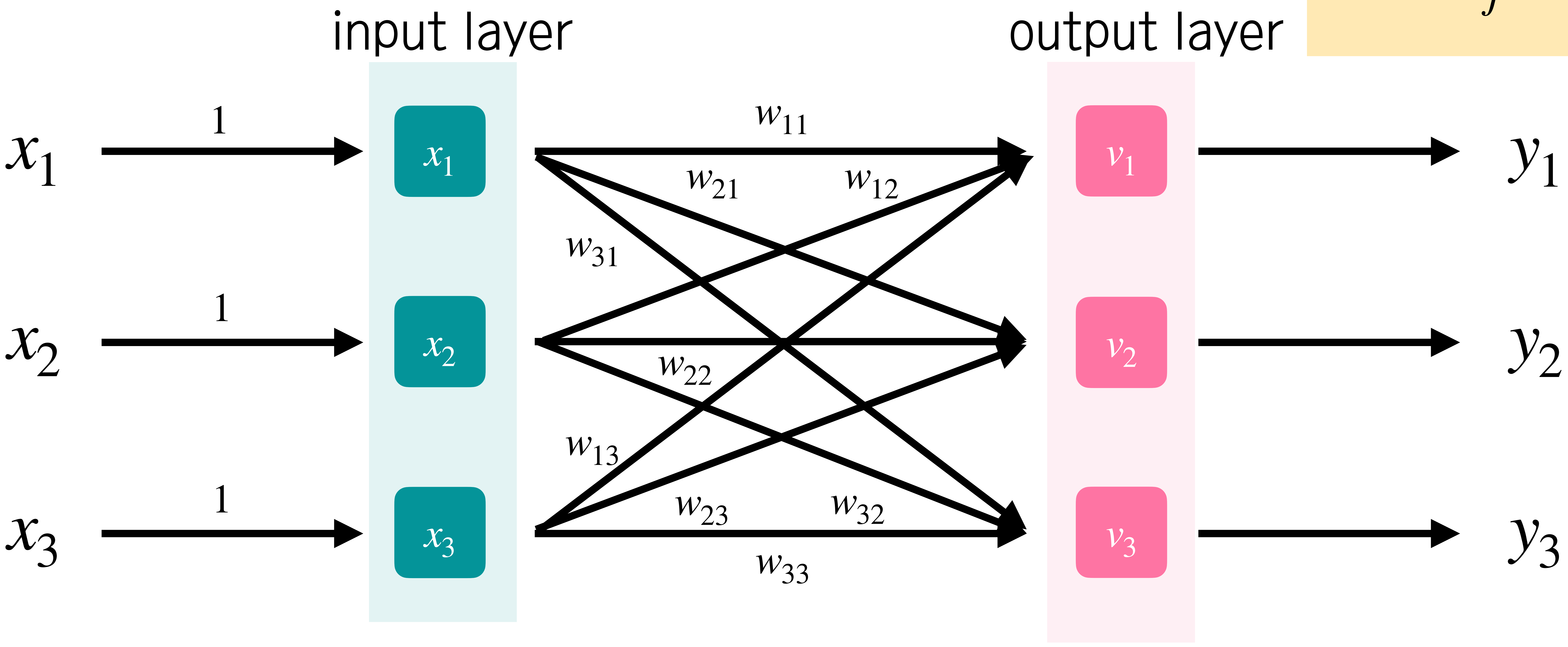


Perceptron

Weight w_{ij} is multiplied to the input signal x_j to the output node i

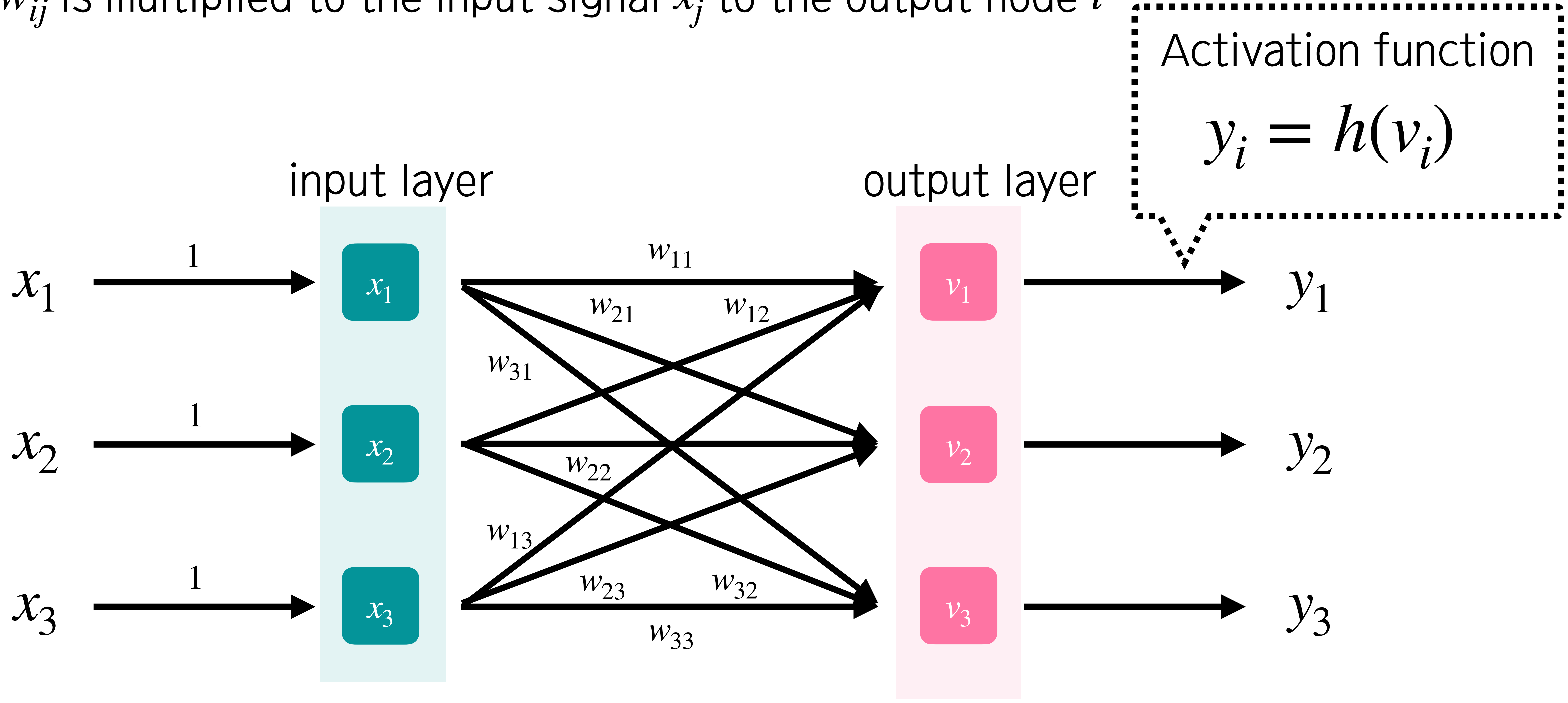
$$v_i = \sum_j w_{ij}x_j + b_i$$

bias
(편향)



Perceptron

Weight w_{ij} is multiplied to the input signal x_j to the output node i



Perceptron learning

We have right answers $\vec{a}$. So, we will update the parameter set to have less error.

Difference $\vec{e} = \vec{a} - \vec{y}$

Error $E = \vec{e} \cdot \vec{e} = \sum_i (\vec{a}_i - \vec{y}_i)^2$

Since $\vec{y}$ depends on w_{ij} set and bias b_i, those parameters are updated by reducing the total error.

Perceptron learning

We have right answers $\vec{a}$. So, we will update the parameter set to have less error.

Difference	$\vec{e} = \vec{a} - \vec{y}$
Error	$E = \vec{e} \cdot \vec{e} = \sum_i (\vec{a}_i - \vec{y}_i)^2$

Since $\vec{y}$ depends on w_{ij} set and bias b_i, those parameters are updated by reducing the total error.

We will use gradient descent method

$$W_{ij} \to W_{ij} - c\frac{\partial E}{\partial W_{ij}}, \;\; b_i \to b_i - c\frac{\partial E}{\partial b_i}$$

Perceptron learning

Heaviside step function has zero differential value, so it is better to use sigmoid function. It is because (1) the output value is bounded from 0 to 1 and (2) differentiation is well defined.

$$h(v) = \frac{1}{1 + e^{-v}}$$

Using chain rule and $\dfrac{\partial h(v)}{\partial v} = h(v)\left[1 - h(v)\right]$

$$W_{ij} \rightarrow W_{ij} + c e_i y_i (1 - y_i) x_j$$

$$b_i \rightarrow b_i + c e_i y_i (1 - y_i)$$

Example – AND operator

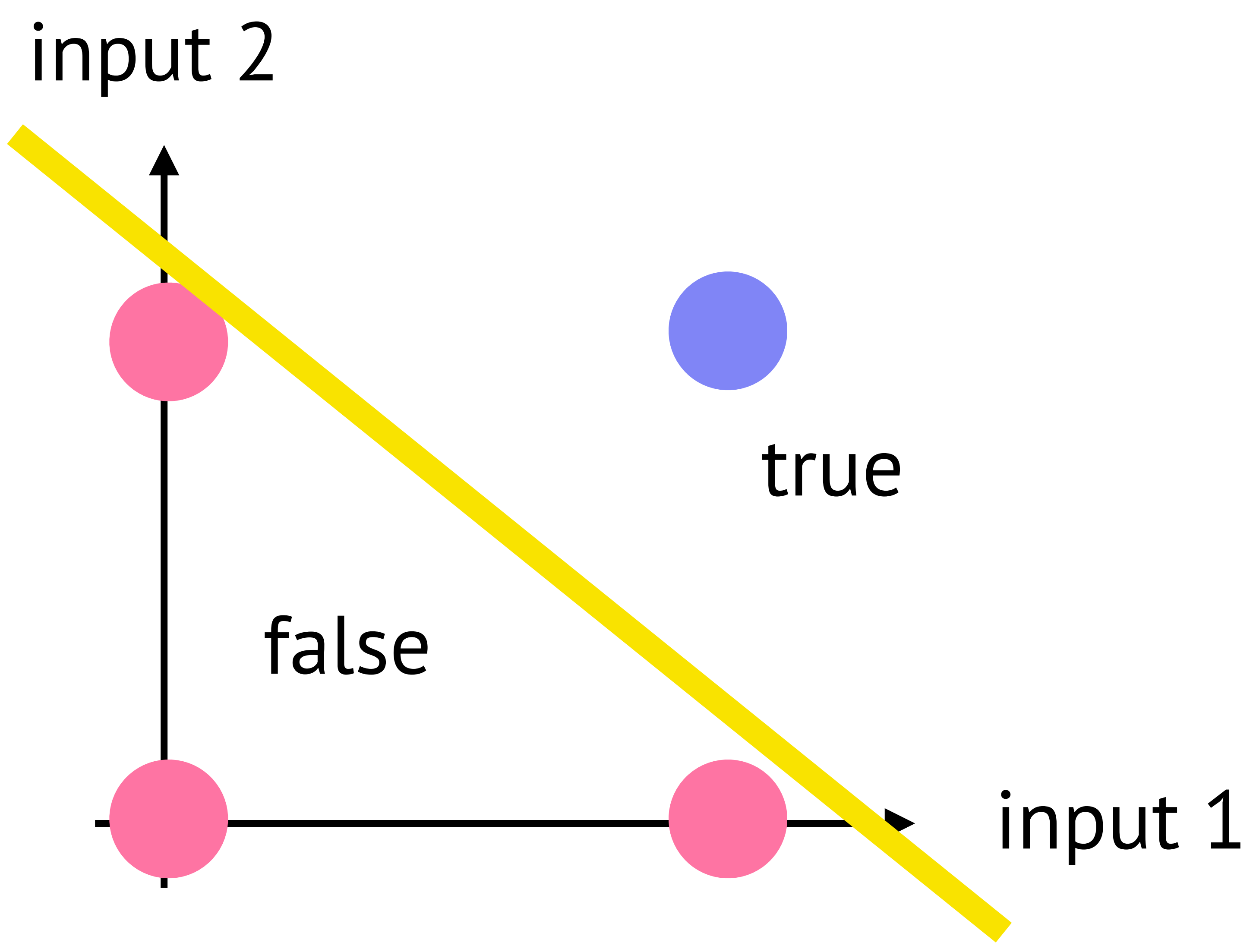
AND operator

input 1	input 2	output
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

Example - AND operator

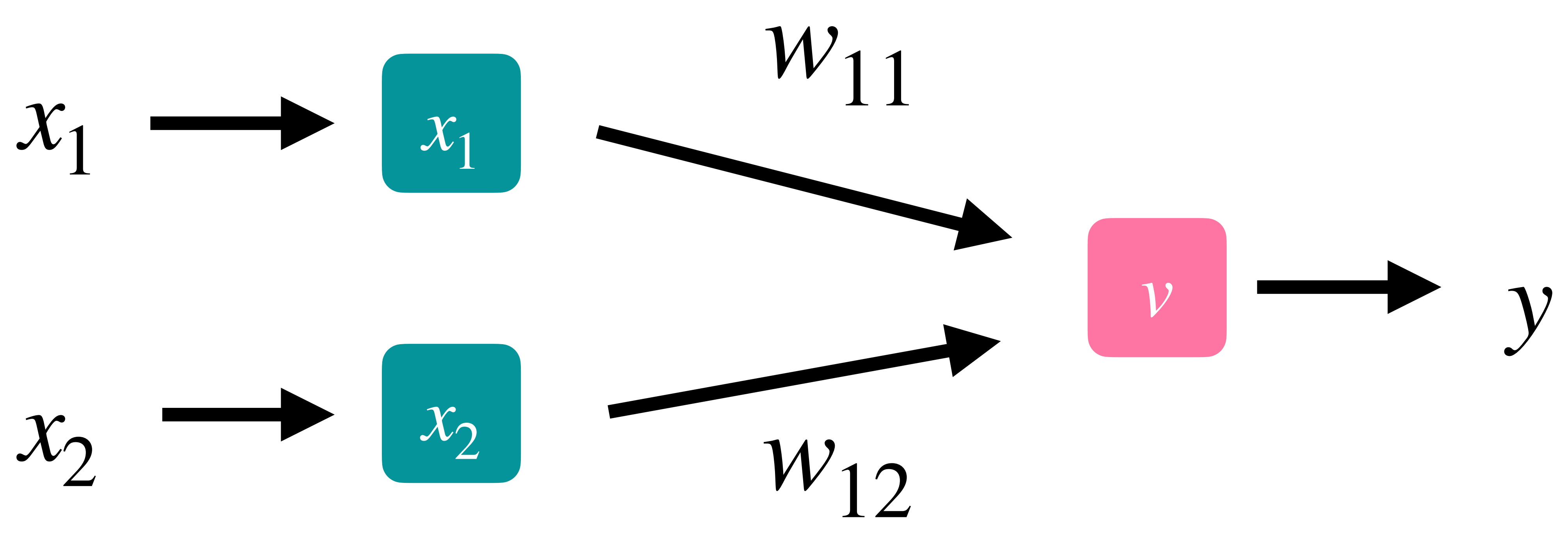
AND operator

input 1	input 2	output
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE



Example - AND operator

Let's find w_{11}, w_{12}, b

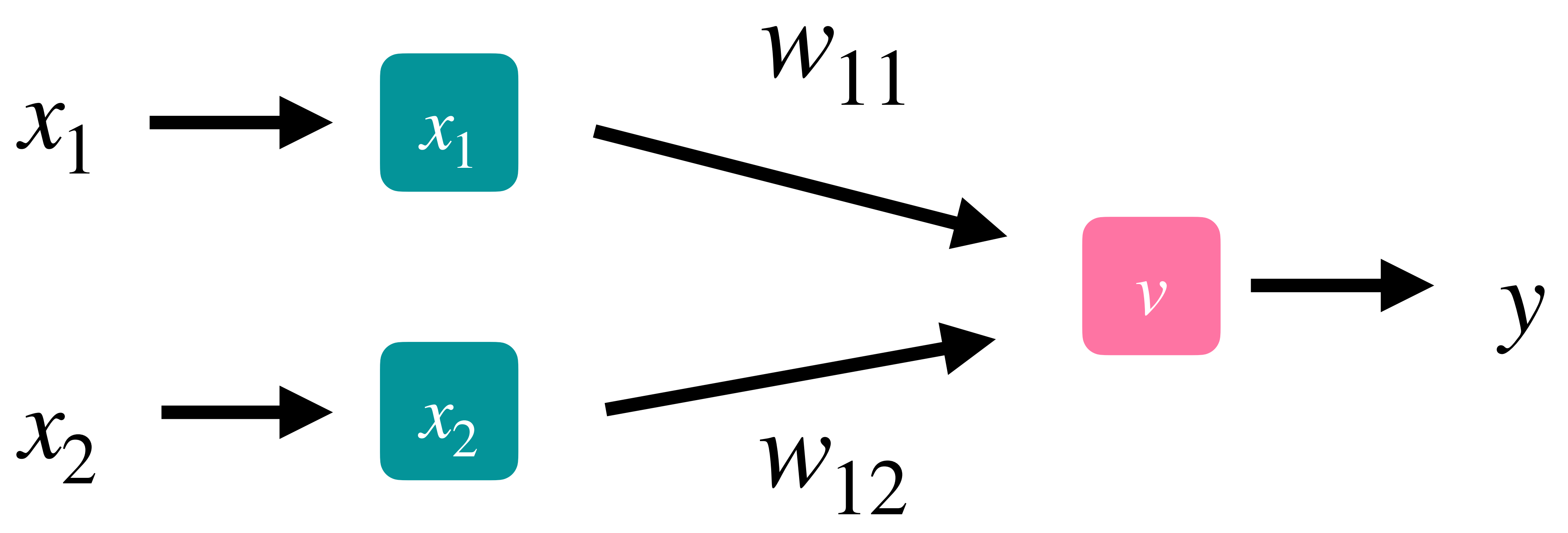


$$v = w_{11}x_1 + w_{12}x_2 + b$$

$$y = h\left(w_{11}x_1 + w_{12}x_2 + b\right)$$

Example - AND operator

Let's find w_{11}, w_{12}, b



$$v = w_{11}x_1 + w_{12}x_2 + b$$

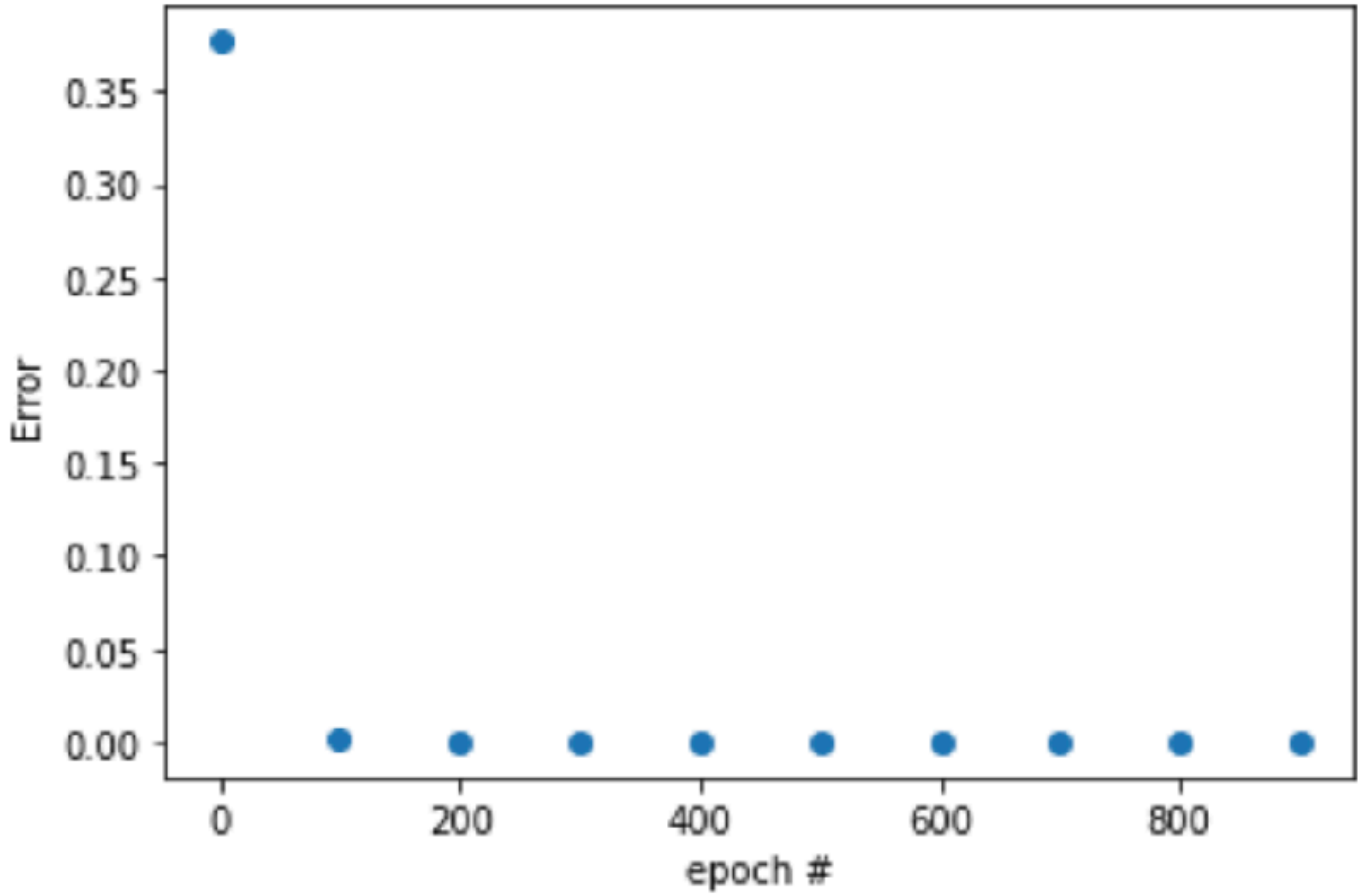
$$y = h\left(w_{11}x_1 + w_{12}x_2 + b\right)$$

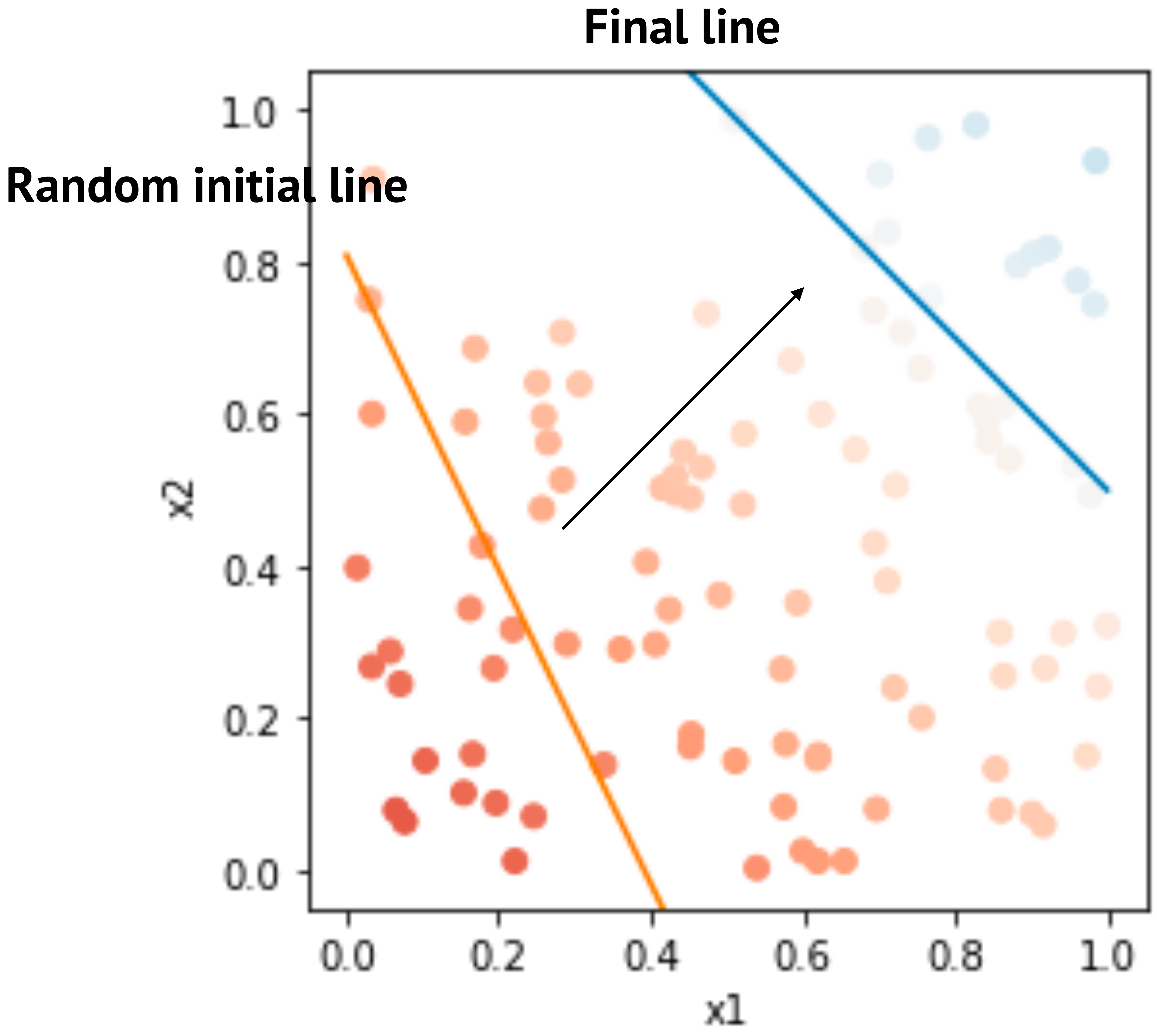
$$W_{1j} \to W_{1j} + cey(1-y)x_j$$

$$b \to b + cey(1-y)$$

Example – AND operator

```
[[1.          1.14430635]] 0.819390771984584
[1 1] 1 0.9033107055678224
[1 0] 0 0.08031337785347427
[0 1] 0 0.08059139391472323
[0 0] 0 0.0008186790293276247
[[1.         1.0008049]] -1.5221991020149175
```





Let's try to make a code for XOR operators

XOR operator

input 1	input 2	output
TRUE	TRUE	FALSE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

Let's try to make a code for XOR operators

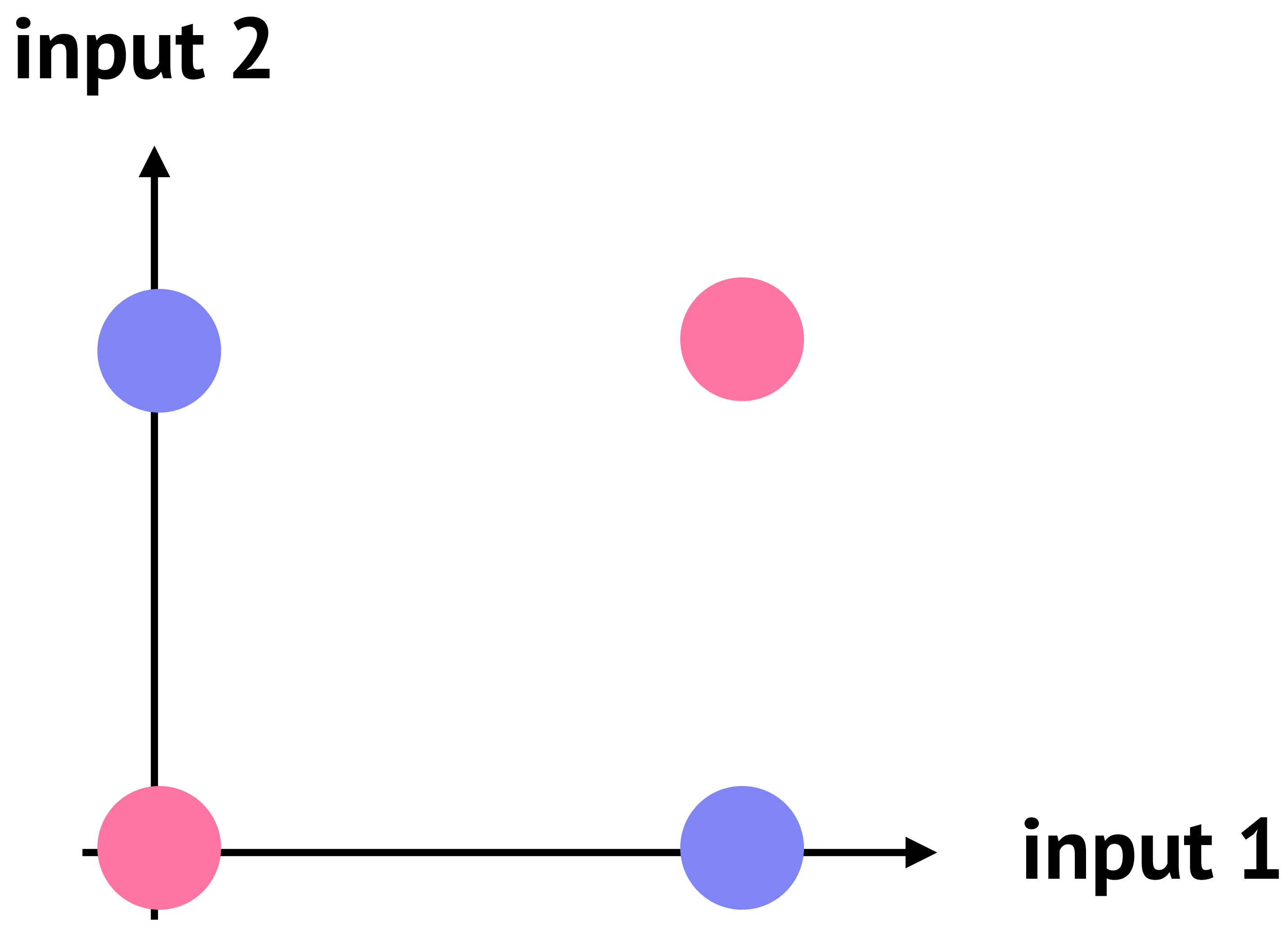
XOR operator

input 1	input 2	output
TRUE	TRUE	FALSE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

Let's try to make a code for XOR operators

XOR operator

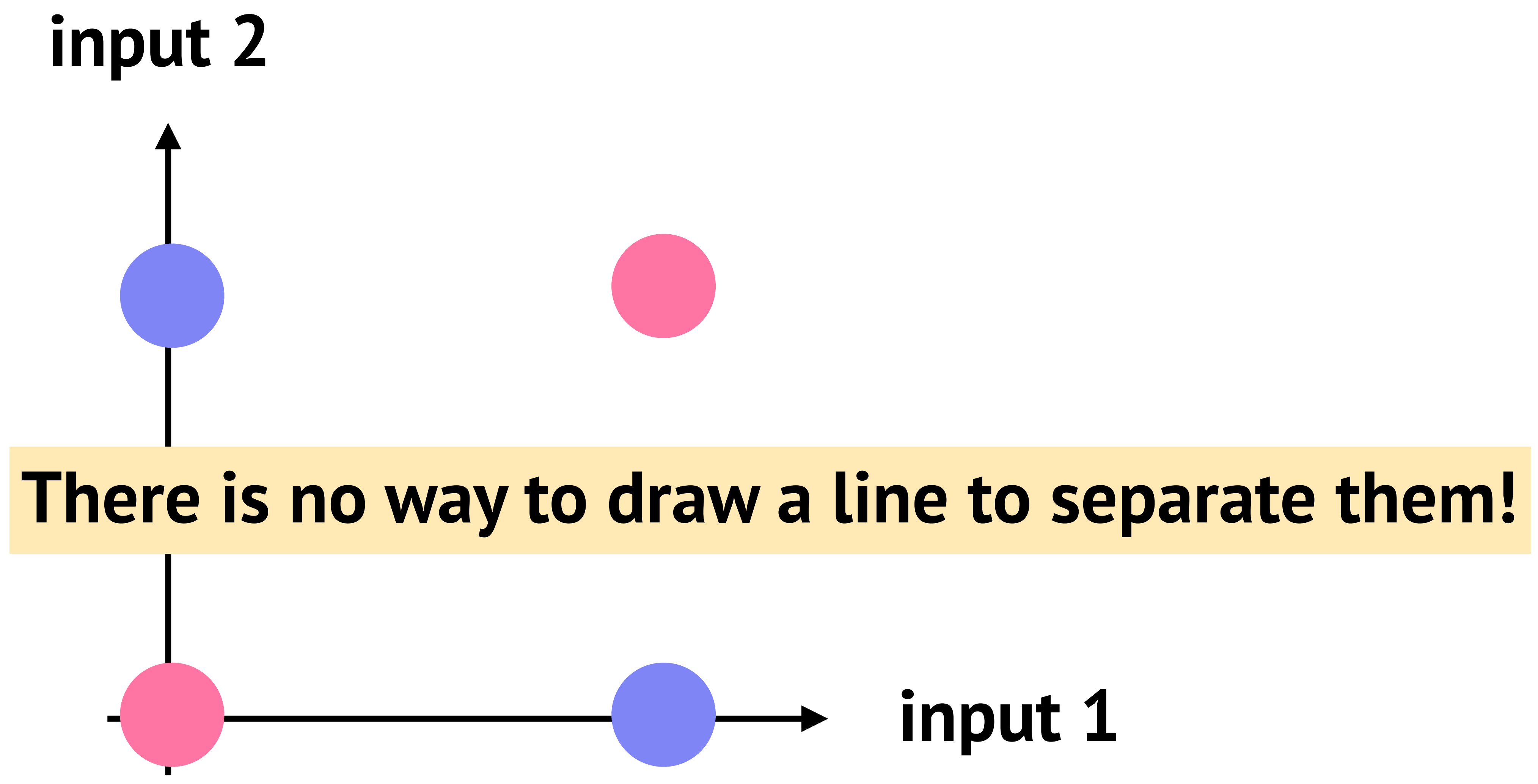
input 1	input 2	output
TRUE	TRUE	FALSE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE



Let's try to make a code for XOR operators

XOR operator

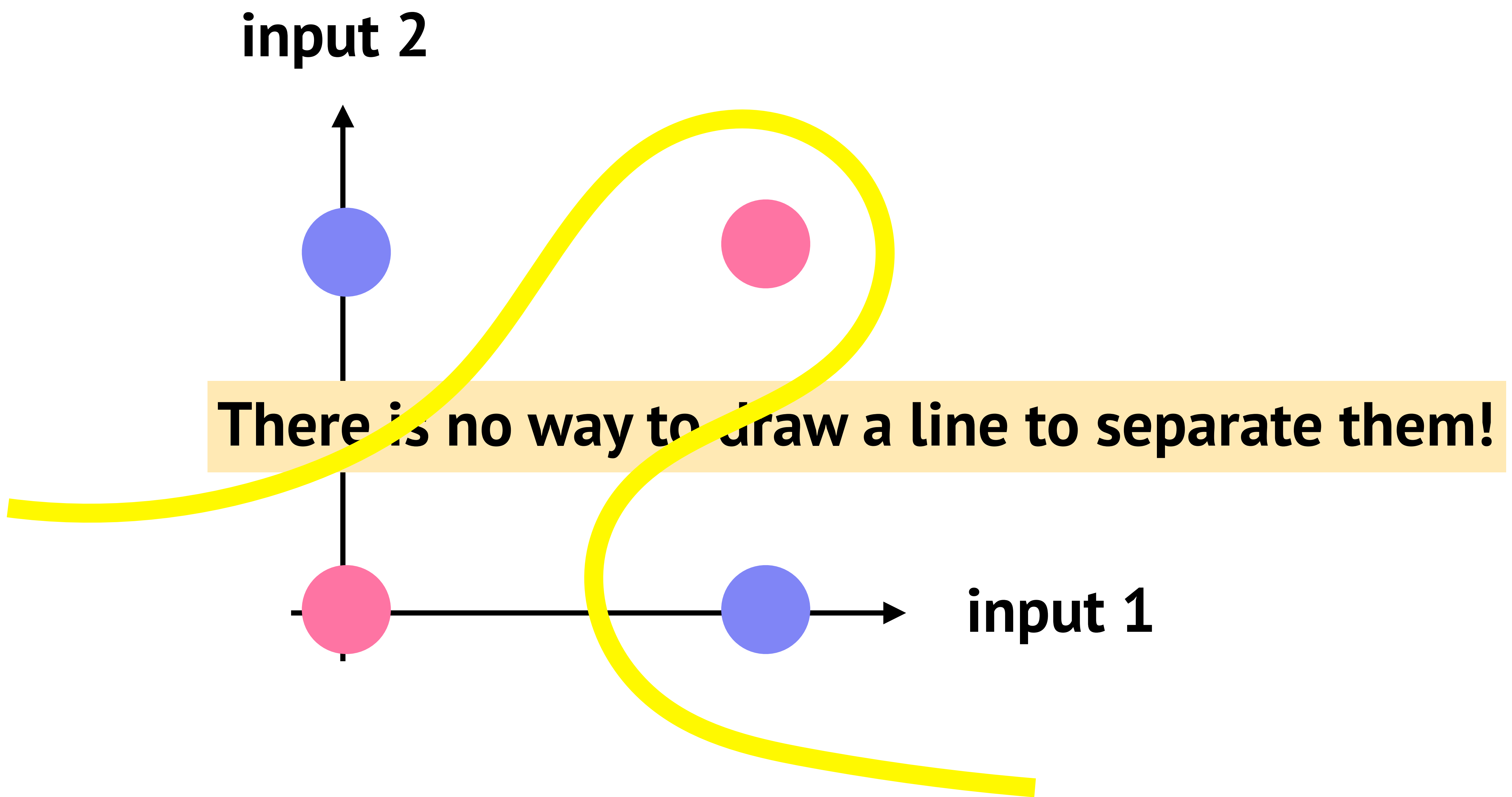
input 1	input 2	output
TRUE	TRUE	FALSE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE



Let's try to make a code for XOR operators

XOR operator

input 1	input 2	output
TRUE	TRUE	FALSE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

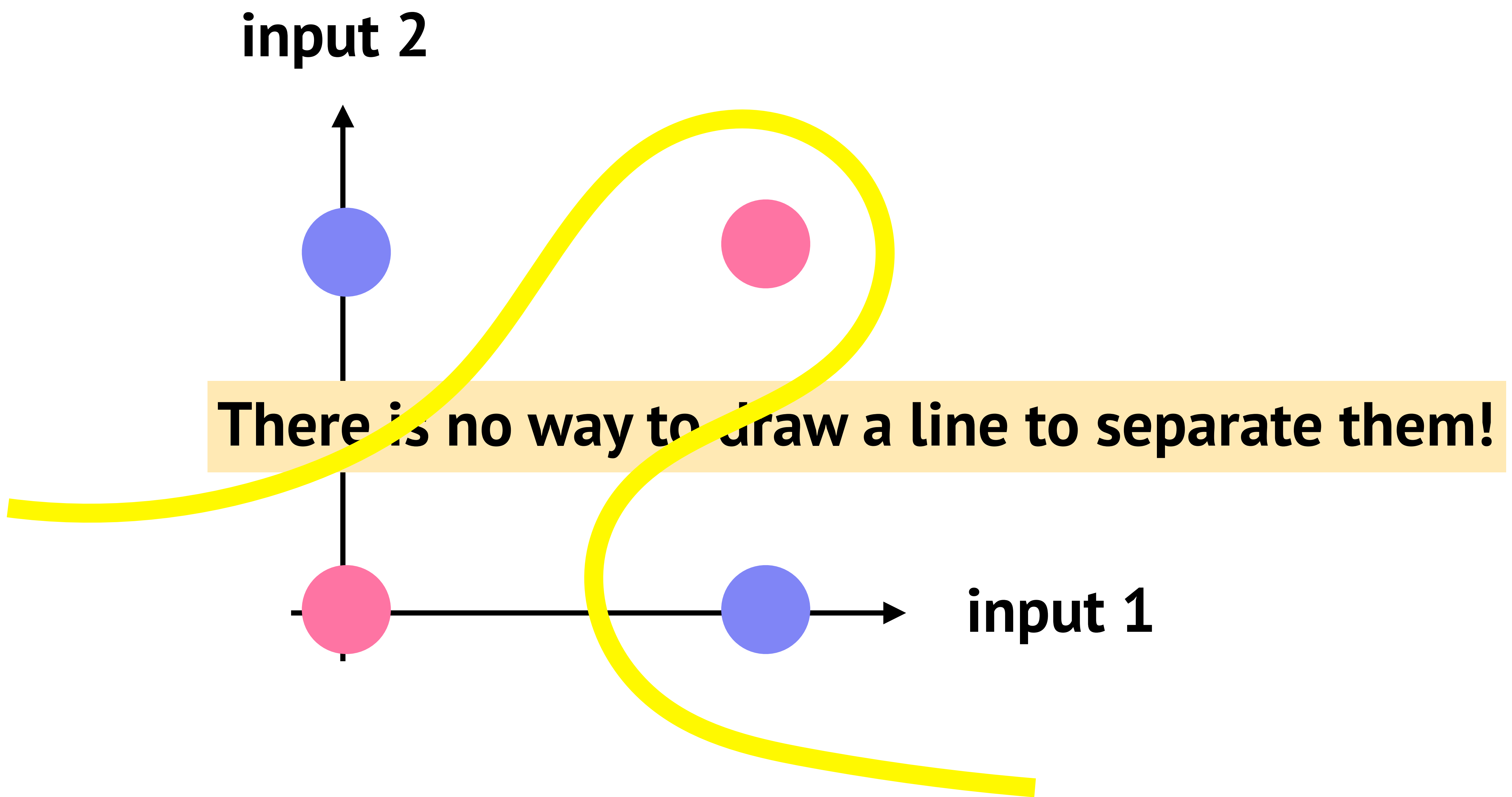


Let's try to make a code for XOR operators

Then, how to overcome?

XOR operator

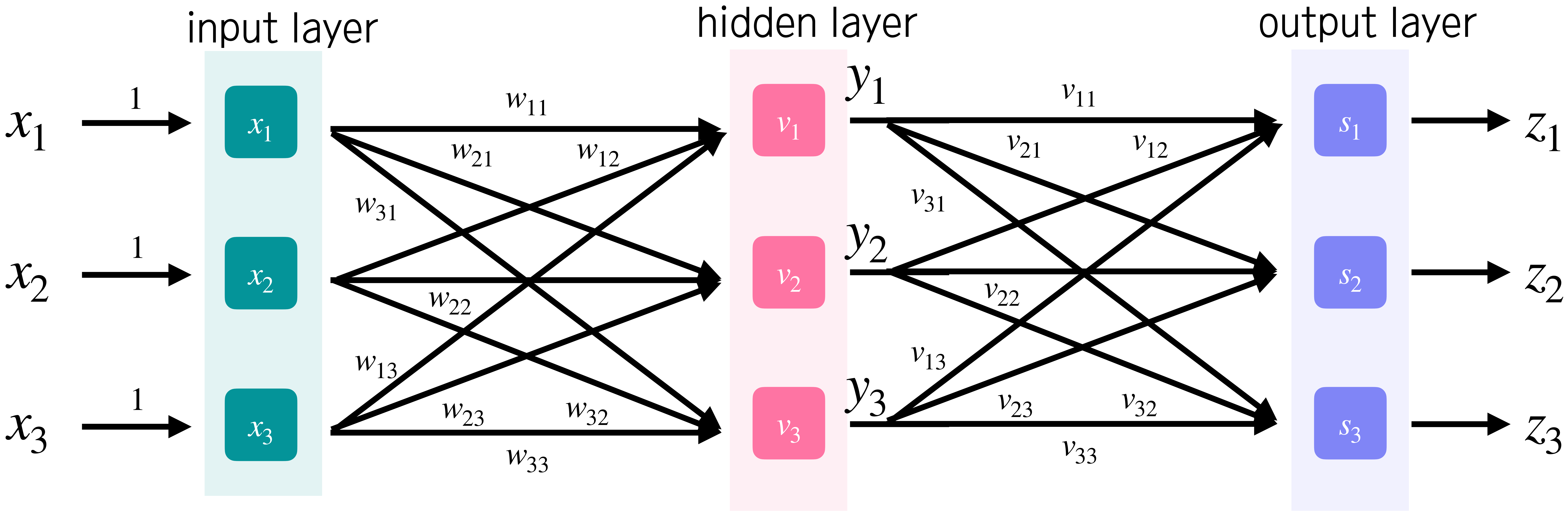
input 1	input 2	output
TRUE	TRUE	FALSE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE



Perceptron with a hidden layer

[https://nnfs.io/mvp/]

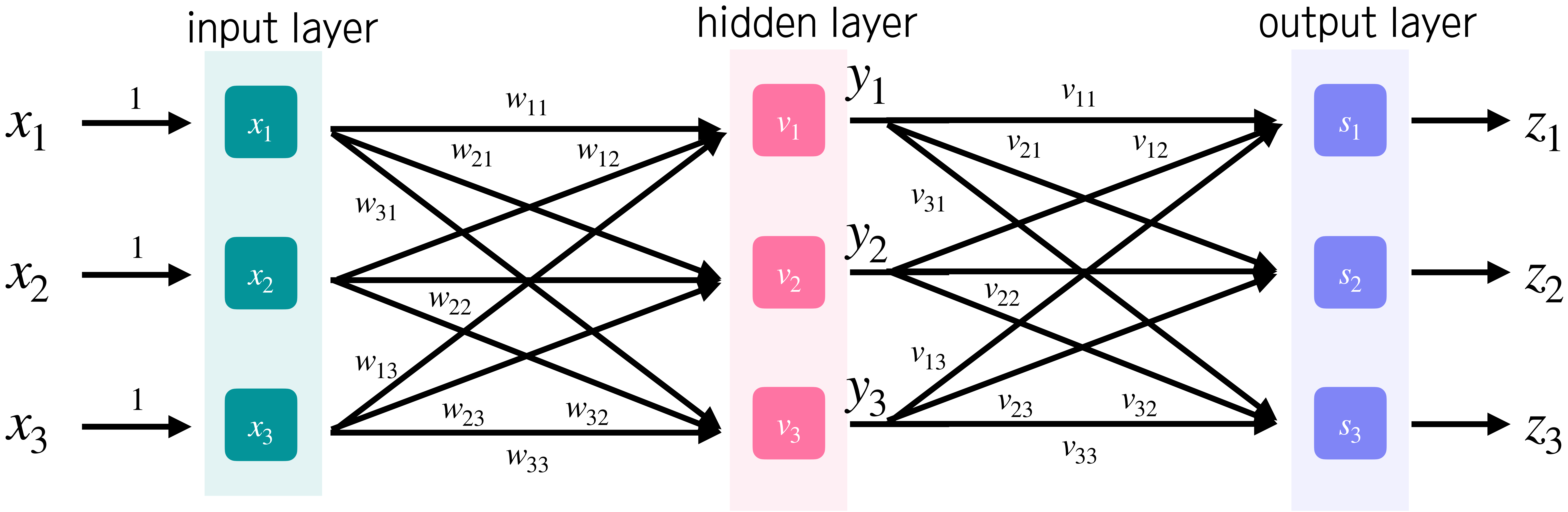
Make the system nonlinear ⇨ adding a hidden layer!
# of nodes in a hidden layer and # of hidden layers are the free parameters.



[https://nnfs.io/mvp/]

Perceptron with a hidden layer

Make the system nonlinear ⇨ adding a hidden layer!
# of nodes in a hidden layer and # of hidden layers are the free parameters.

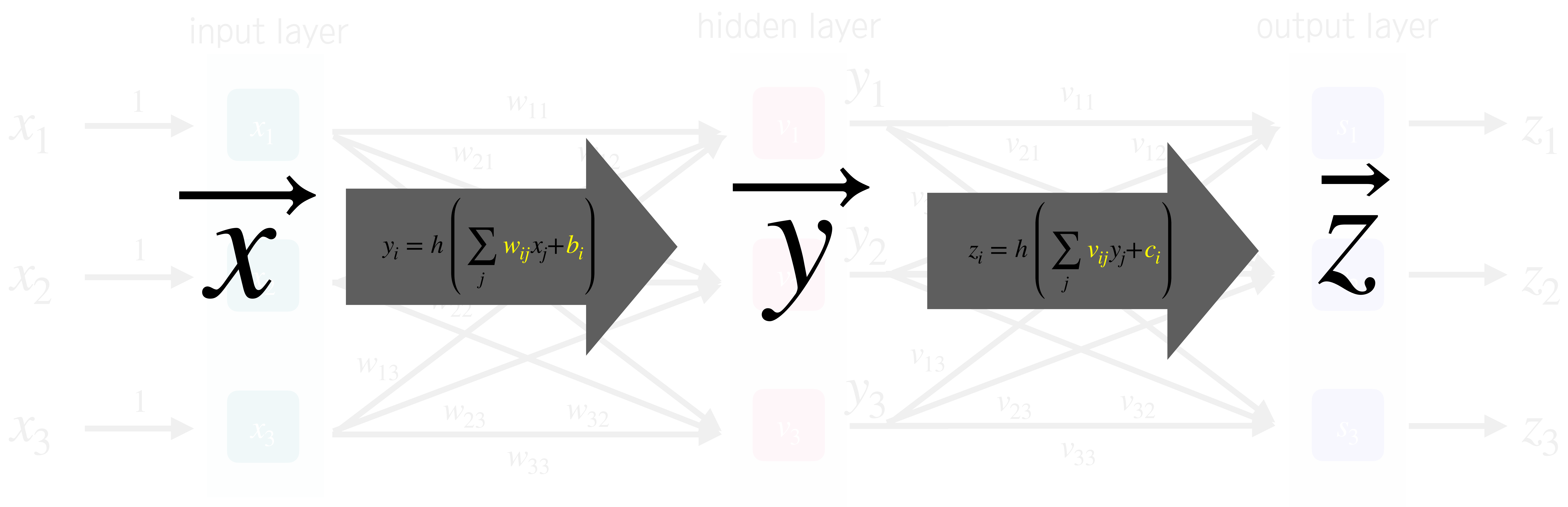


$$v_i = \sum_j w_{ij} x_j + b_i$$
$$y_i = h(v_i)$$

$$s_i = \sum_j v_{ij} y_j + c_i$$
$$z_i = h(s_i)$$

Perceptron with a hidden layer

Make the system nonlinear ⇨ adding a hidden layer!
# of nodes in a hidden layer and # of hidden layers are the free parameters.



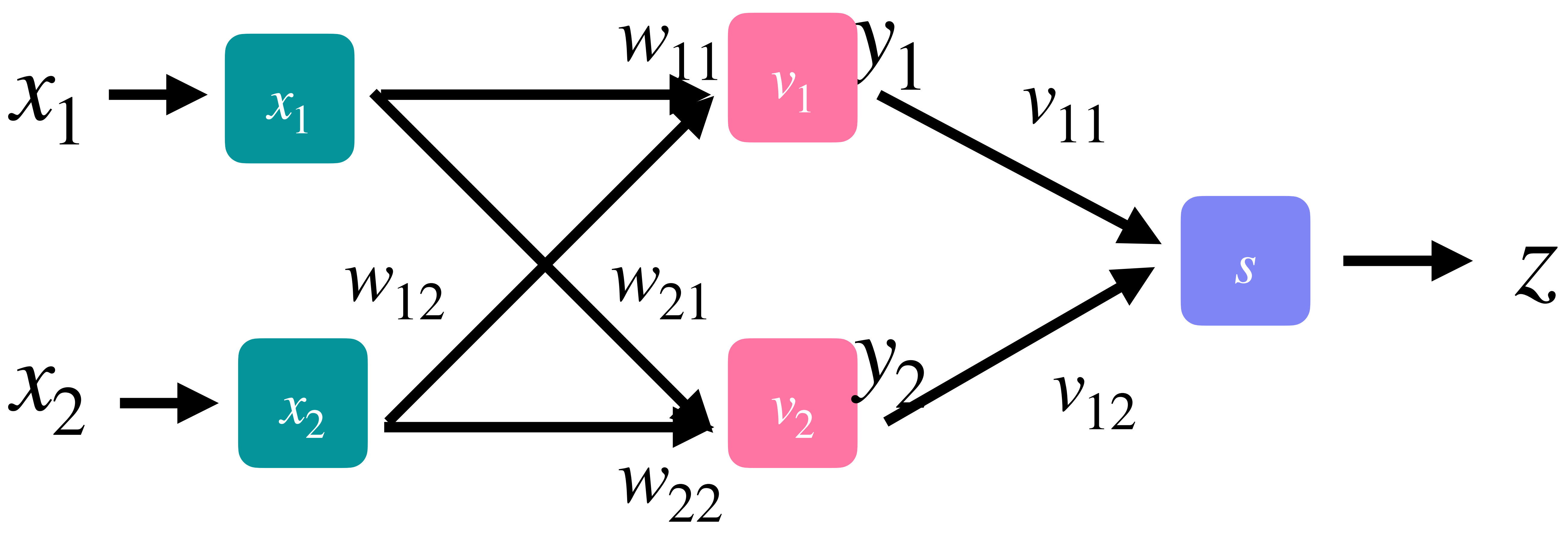
$$W_{ij} \to W_{ij} + \alpha y_i(1 - y_i)x_j(V^T)_i$$

$$b_i \to b_i + \alpha y_i(1 - y_i)(V^T)_i$$

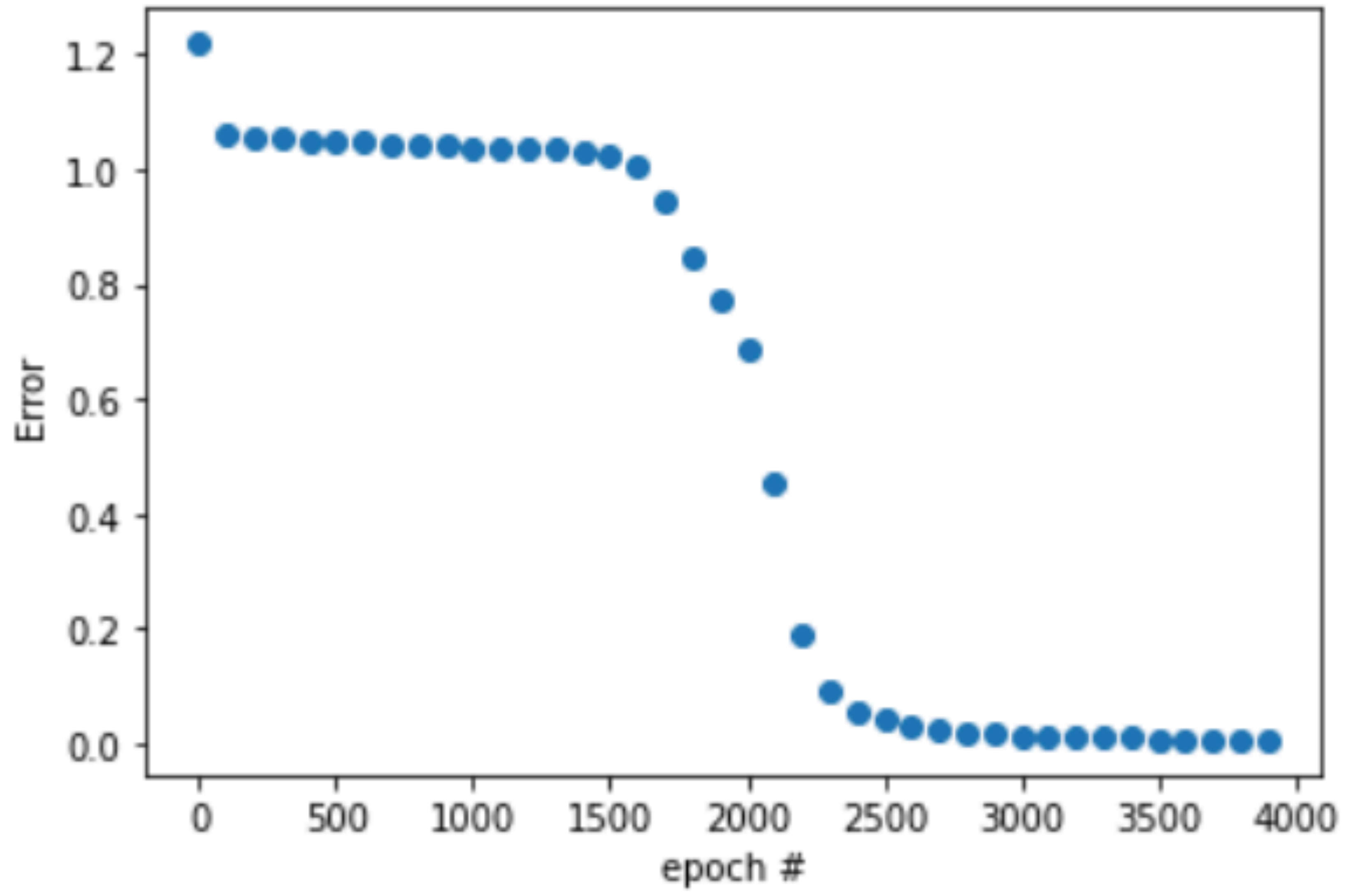
$$V_{ij} \to V_{ij} + \alpha e_i z_i(1 - z_i)y_j$$

$$c_i \to c_i + \alpha e_i z_i(1 - z_i)$$

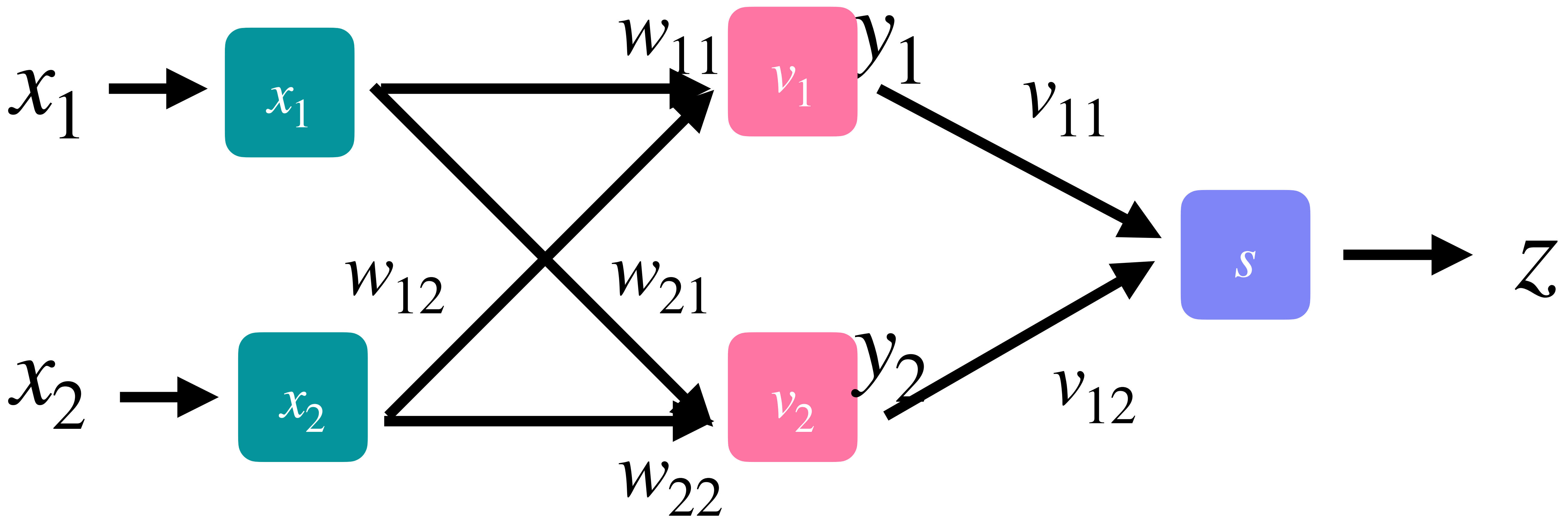
Example - XOR operator



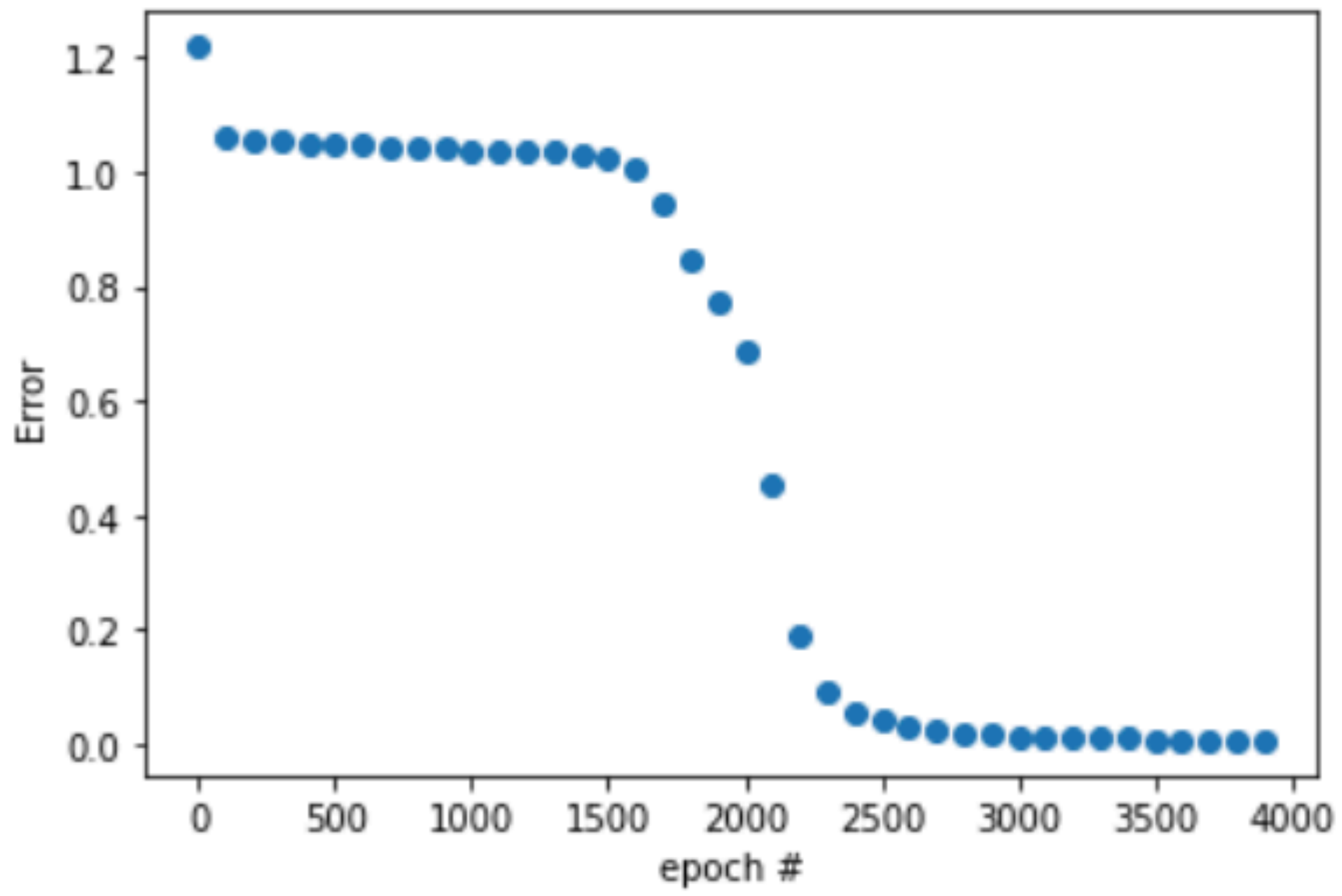

```
0 [1 1] 0 [[0.03600886]]
1 [1 0] 1 [[0.95303802]]
2 [0 1] 1 [[0.96323028]]
3 [0 0] 0 [[0.04028346]]
```

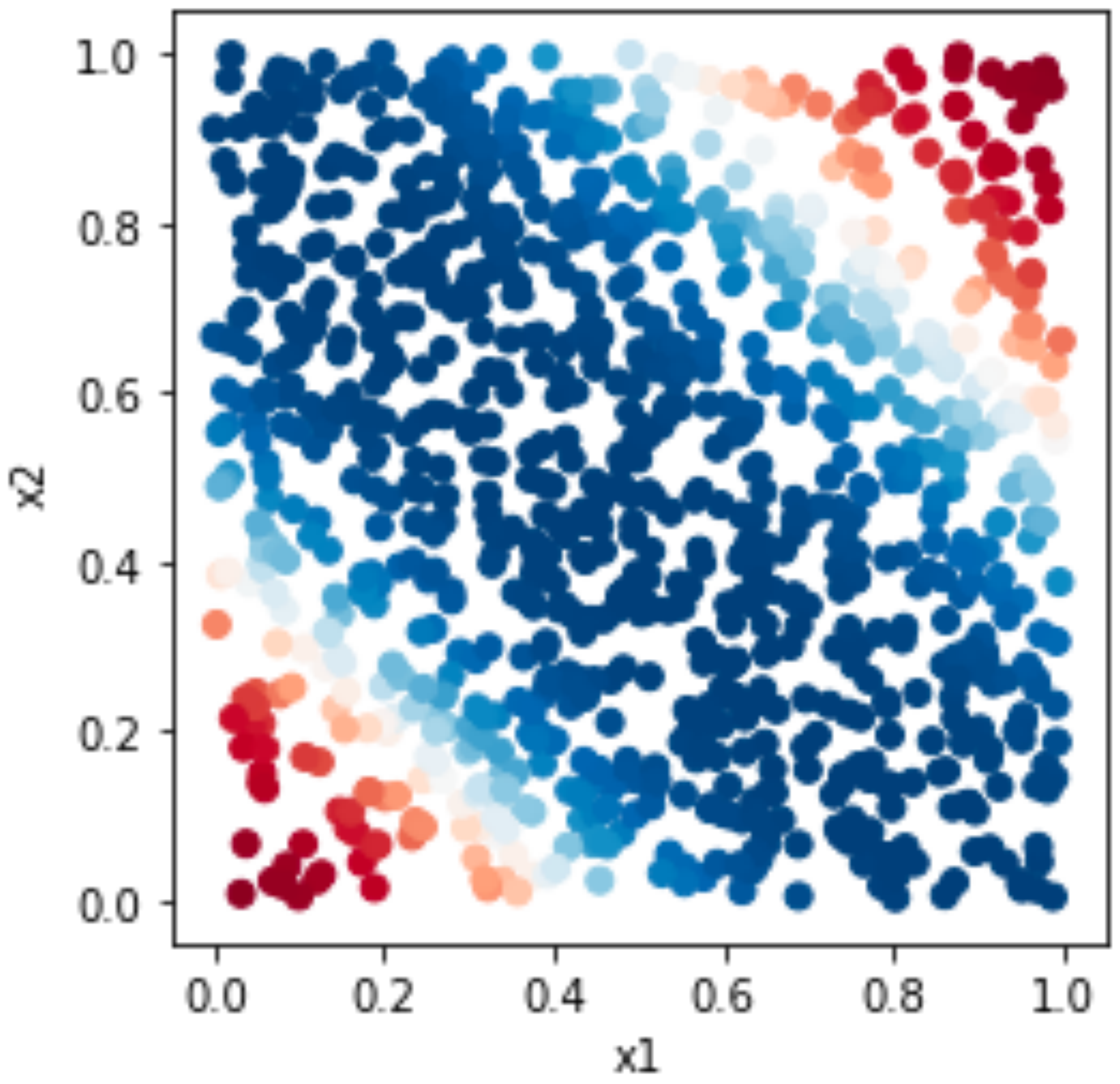


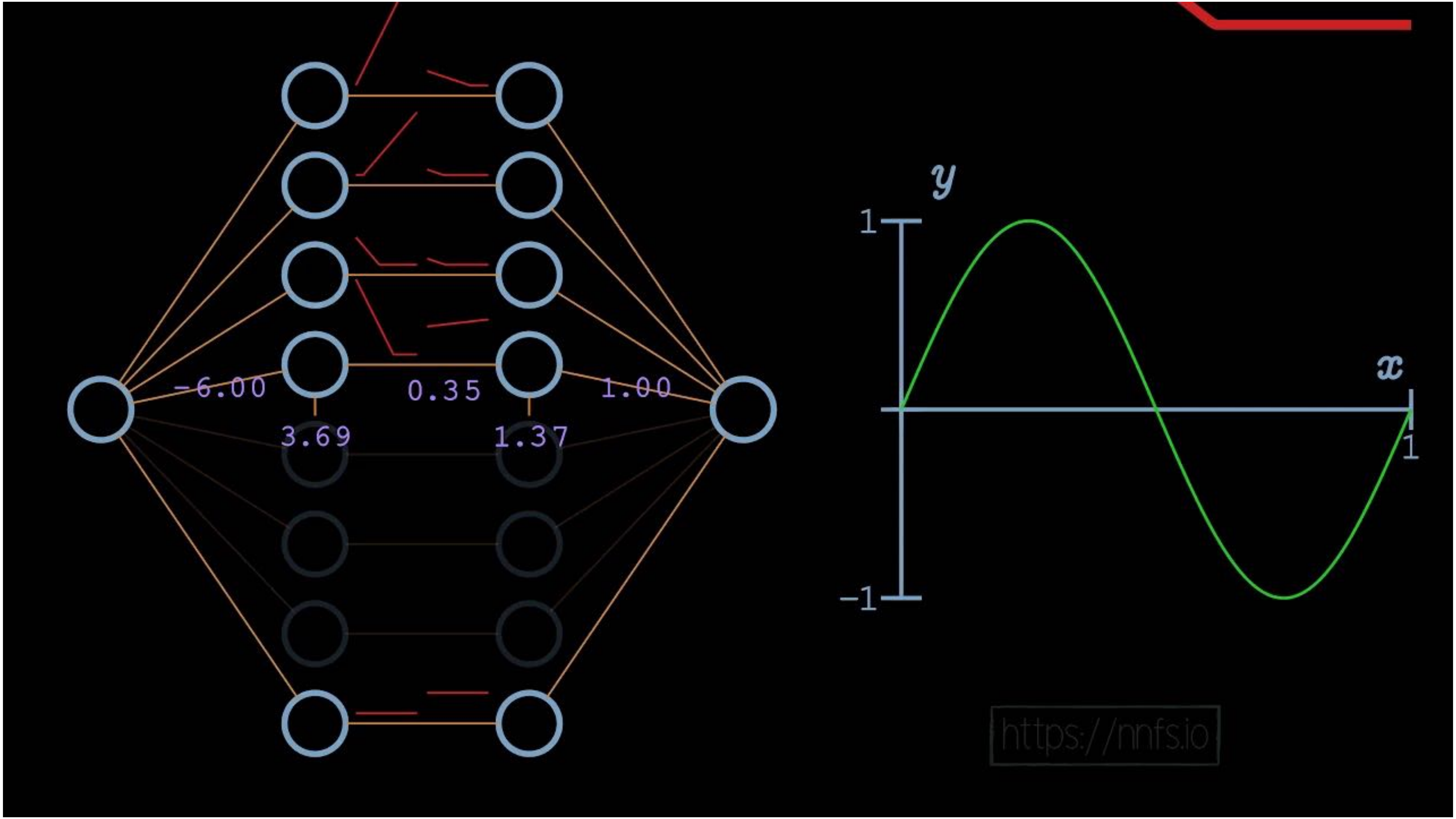
Example – XOR operator

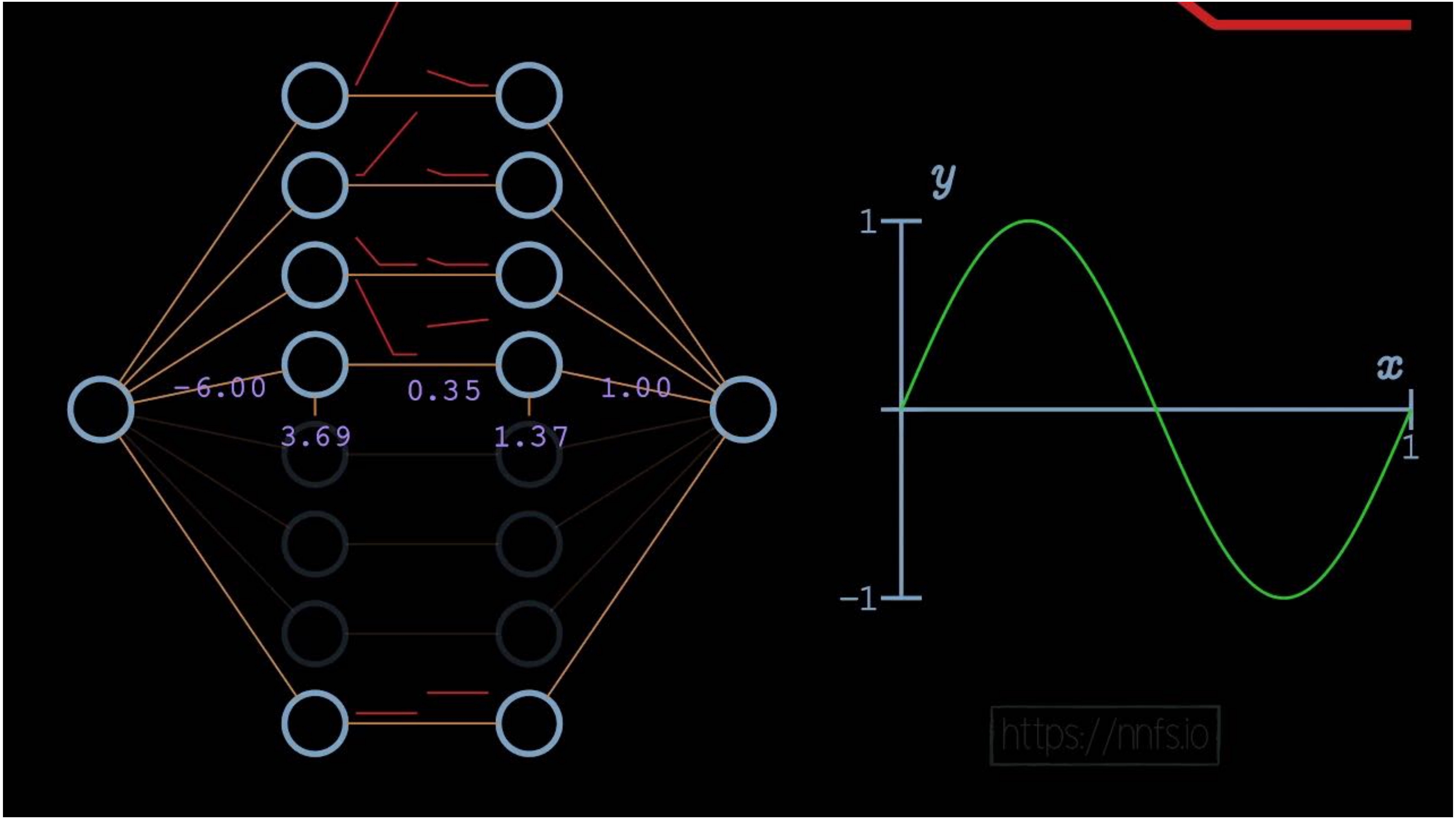



```
0 [1 1] 0 [[0.03600886]]
1 [1 0] 1 [[0.95303802]]
2 [0 1] 1 [[0.96323028]]
3 [0 0] 0 [[0.04028346]]
```





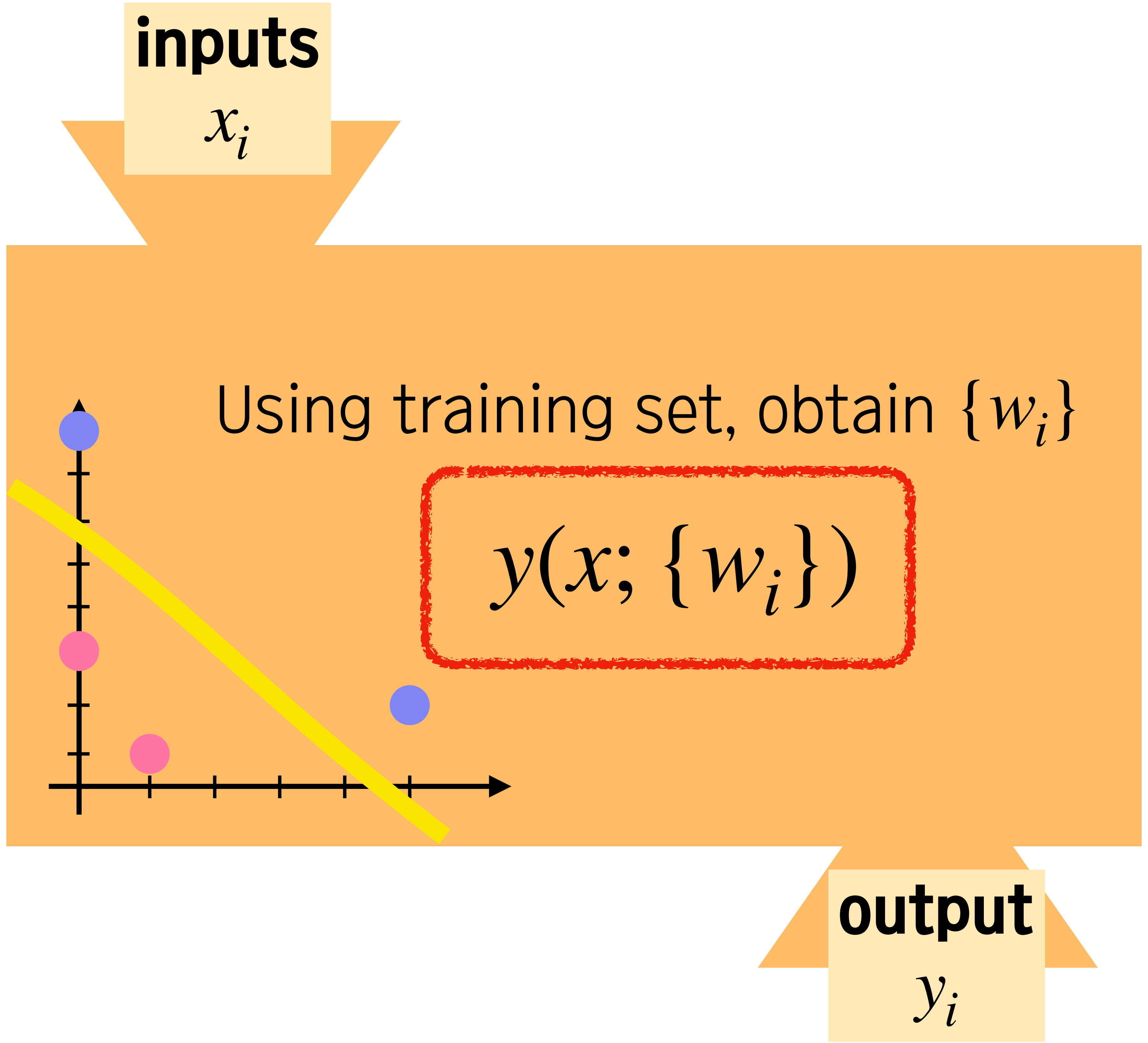

-6.00
0.35
1.00
3.69
1.37
y
1
-1
x
1
https://nnfs.io



-6.00
0.35
1.00
3.69
1.37
y
1
-1
x
1
https://nnfs.io

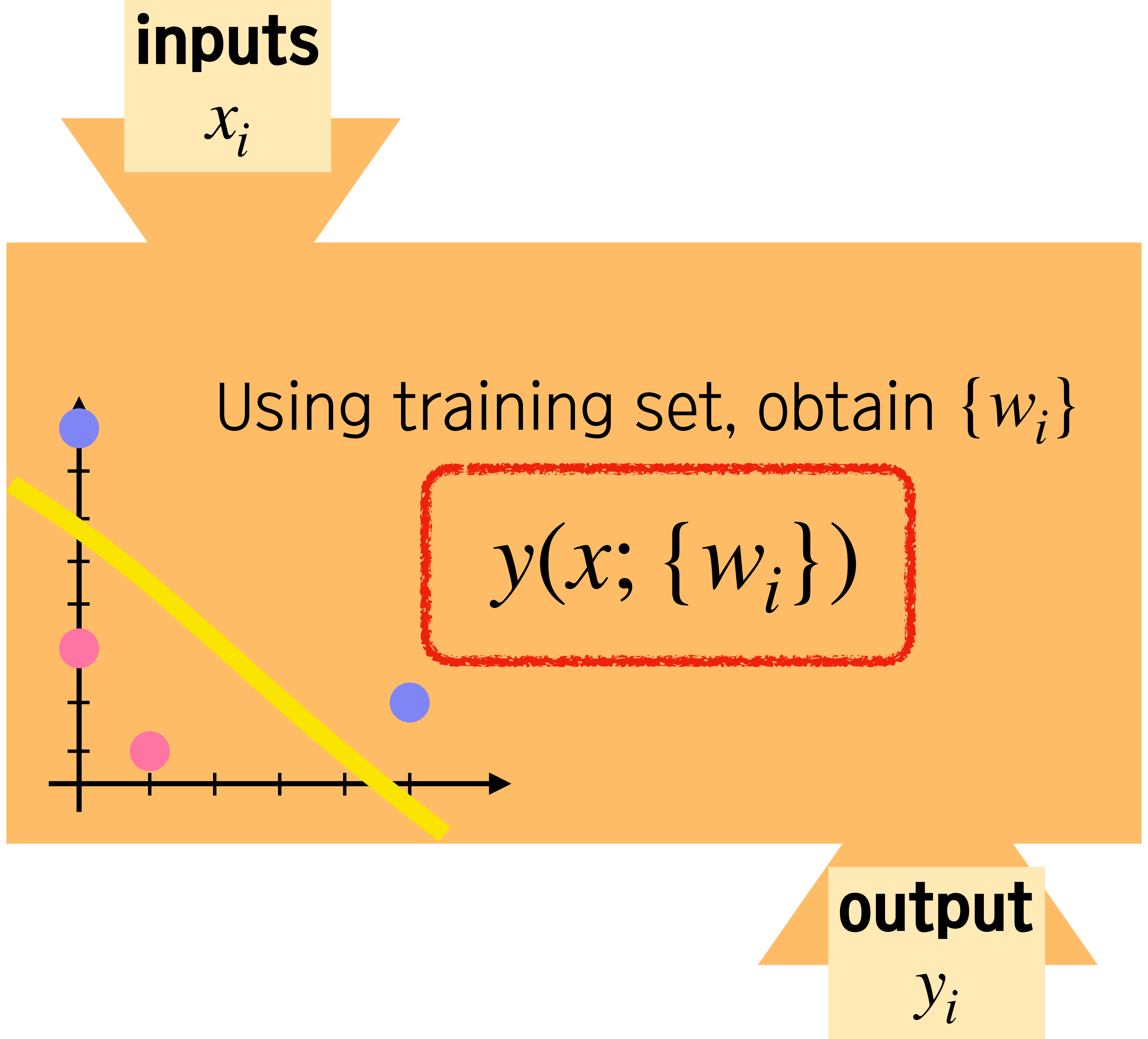
Summary

Classification (or pattern recognization)
is finding a function!

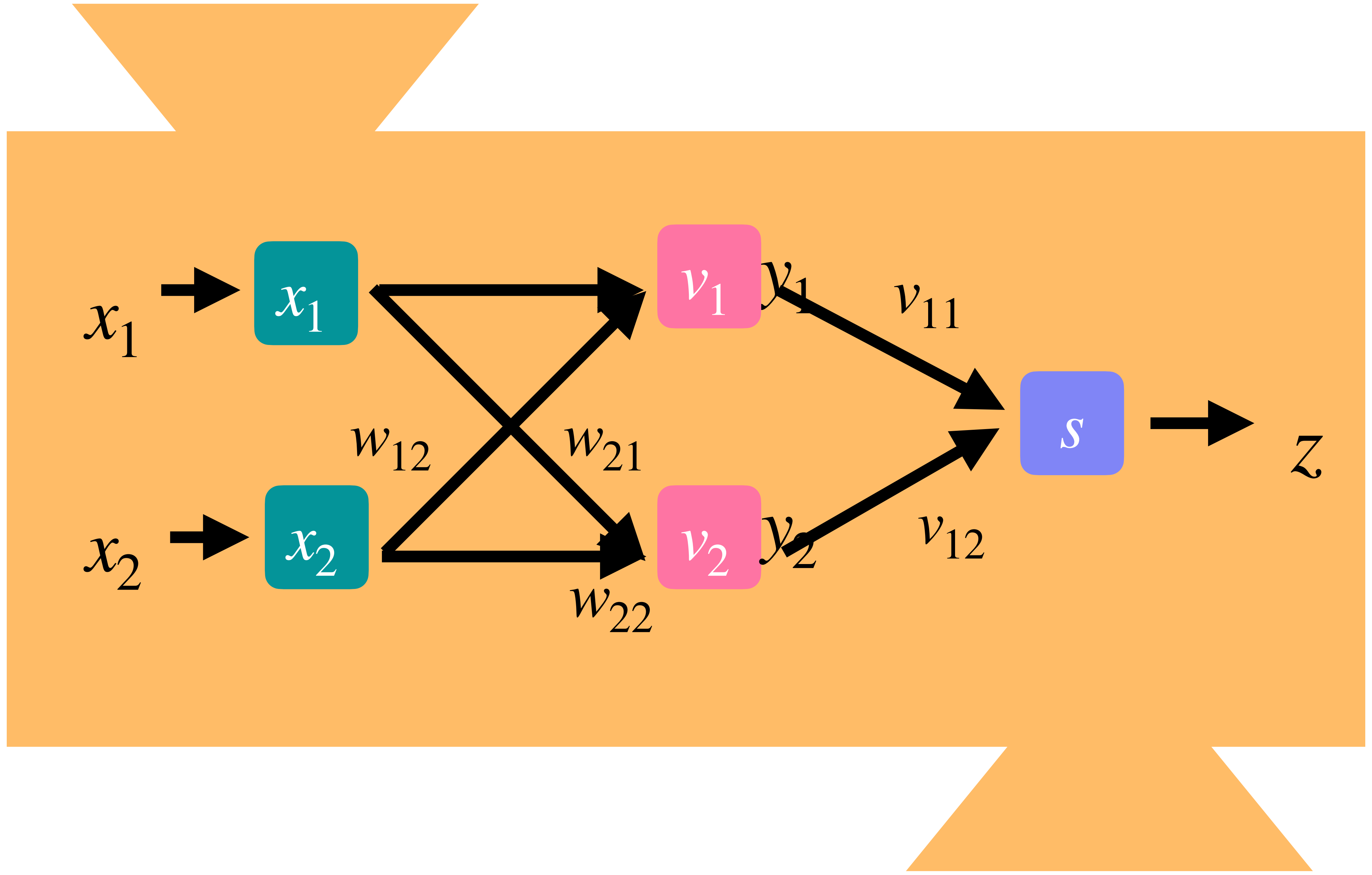


Summary

Classification (or pattern recognization) is finding a function!



Perceptron is one of the models to find such mapping.



Appendix

Example – AND operator

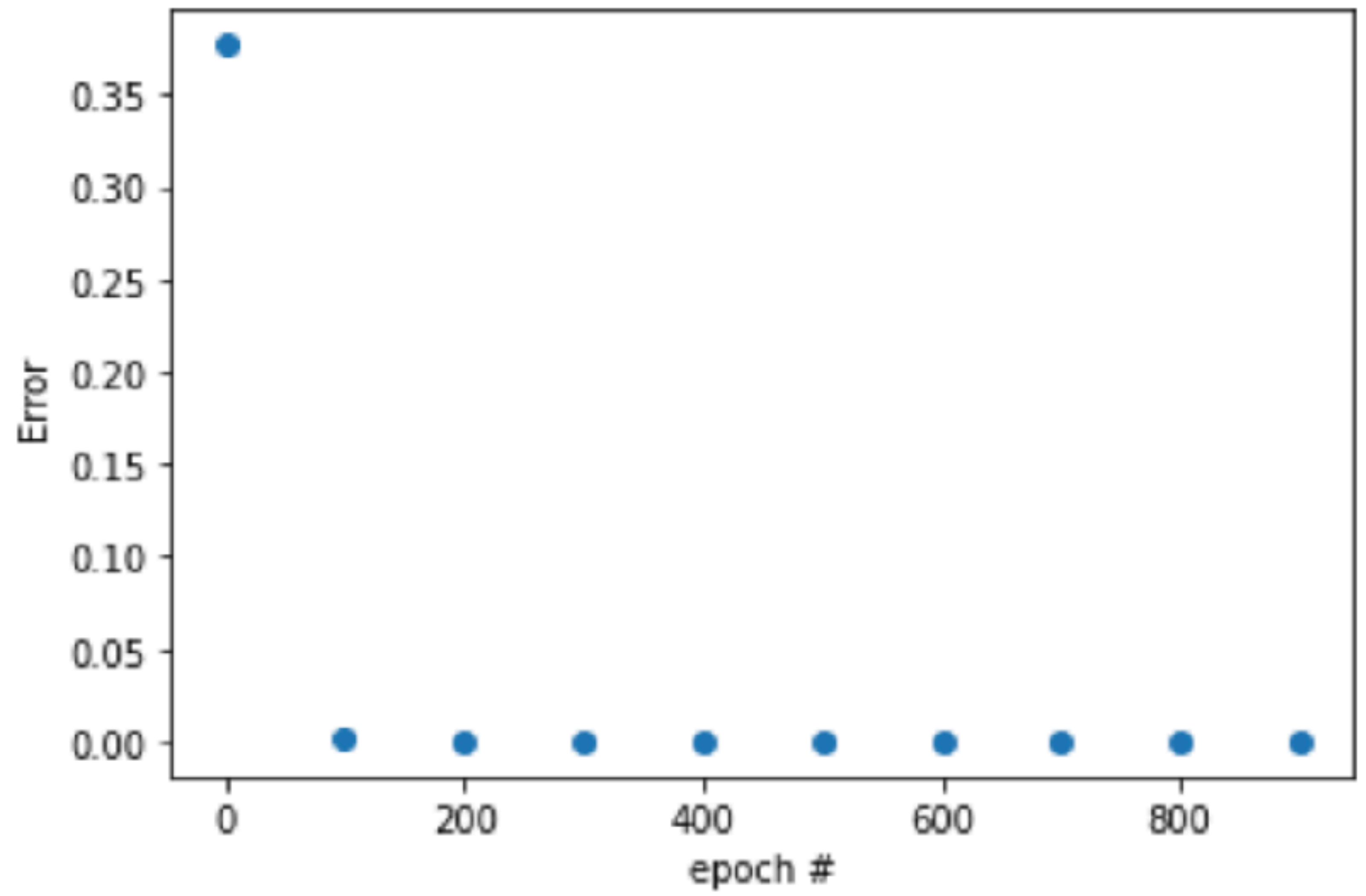
```
import numpy as np, matplotlib.pyplot as plt

N = 4 # N = number of training data
Nep = 1000 # number of epoch
c = 0.5 # learning rate

def h(x):
    return 1.0/(1.0 + np.exp(-x))

x_train = np.array( [[1,1], [1,0], [0,1], [0,0]] ) # training set
a_train = np.array( [1, 0, 0 ,0]) # solution
e = np.zeros(4)
W = np.random.random( (1, 2) ) # random weight
b = np.random.rand()
```

Example – AND operator

```python
for ep in range(Nep):
    for n in range(N):
        x = x_train[n]
        a = a_train[n]
        v = np.sum(W*x) + b
        y = h(v)
        e = a - y
        W = W + c*e*y*(1-y)*x
        b = b + c*e*y*(1-y)
    if (ep % 100 == 0):
        xarr.append(ep)
        yarr.append(e*e)
```

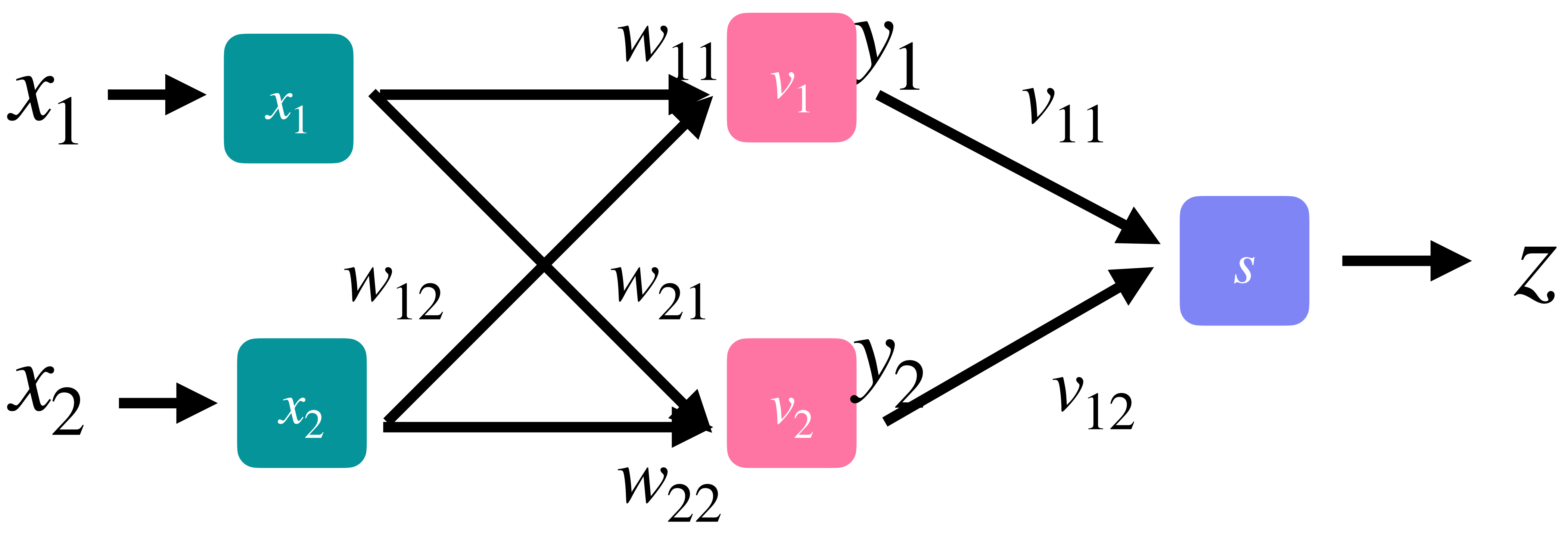
```python
for n in range(N):
    v = np.sum(W*x_train[n]) + b
    y = h(v)
    print(x_train[n], a_train[n], y)

print(W/W[0,0], b/W[0,0])
plt.plot(xarr,yarr, 'o')
plt.xlabel("epoch #")
plt.ylabel("Error")
plt.show()
```

```
[[1.         1.14430635]] 0.819390771984584
[1 1] 1 0.9033107055678224
[1 0] 0 0.08031337785347427
[0 1] 0 0.08059139391472323
[0 0] 0 0.0008186790293276247
[[1.        1.0008049]] -1.5221991020149175
```

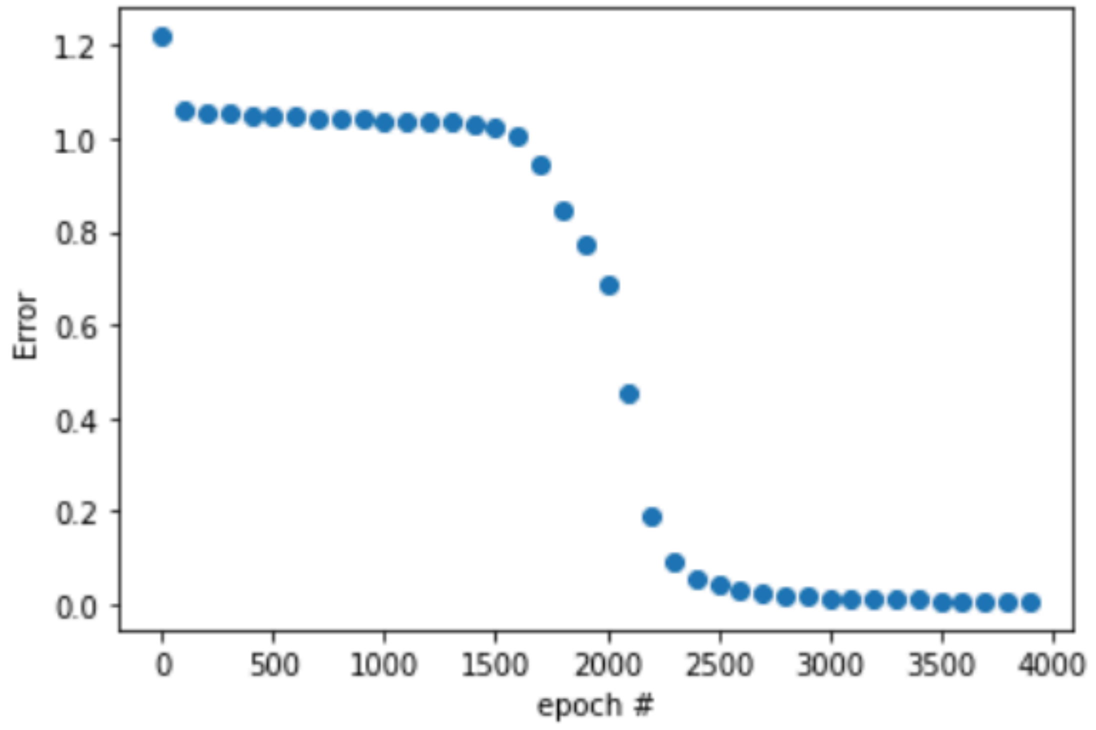


Example - XOR operator

```python
N = 4 # number of traning set
Nep = 4000 # number of epoch
alpha = 0.5 # learning rate

def h(x):
    return 1.0/(1.0 + np.exp(-x))

x_train = np.array( [[1,1], [1,0], [0,1], [0,0]] ) # training set
d_train = np.array( [0, 1, 1 ,0] ) # right answer

W = np.random.random( (2, 2) ) # random weights
b = np.random.random( (2, 1) )

V = np.random.random( (1, 2) ) # random weights
c = np.random.rand()

xarr = []; yarr = []

for ep in range(Nep):
    sume = 0
    for n in range(N):
        x = np.reshape(x_train[n], (2,1))
        d = d_train[n]

        v = W@x + b; y = h(v)
        s = V@y + c; z = h(s)
        e = d - z
        sume += np.ndarray.item(e*e)

        delta = z*(1-z)*e
        e1 = V.T @ delta
        epsil = y*(1-y)*e1

        V += alpha*delta*y.T; c += alpha*delta
        W += alpha*epsil*x.T; b += alpha*epsil

    if (ep % 100 == 0):
        xarr.append(ep); yarr.append(sume)
```

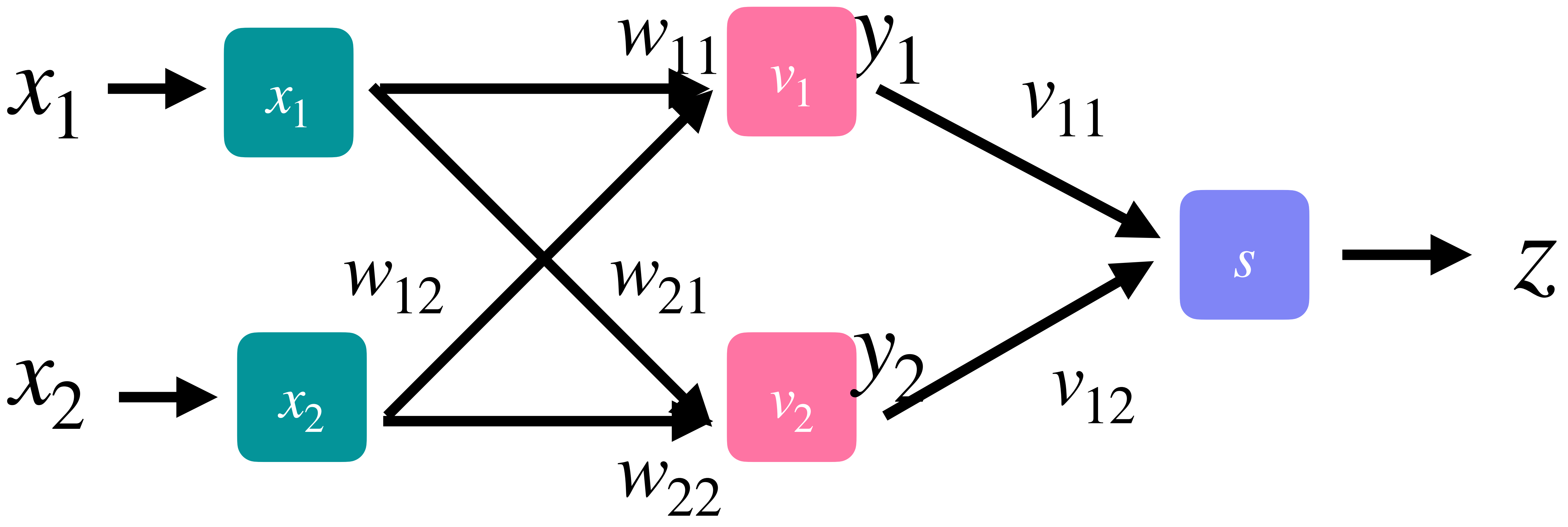



```
0 [1 1] 0 [[0.03600886]]
1 [1 0] 1 [[0.95303802]]
2 [0 1] 1 [[0.96323028]]
3 [0 0] 0 [[0.04028346]]
```

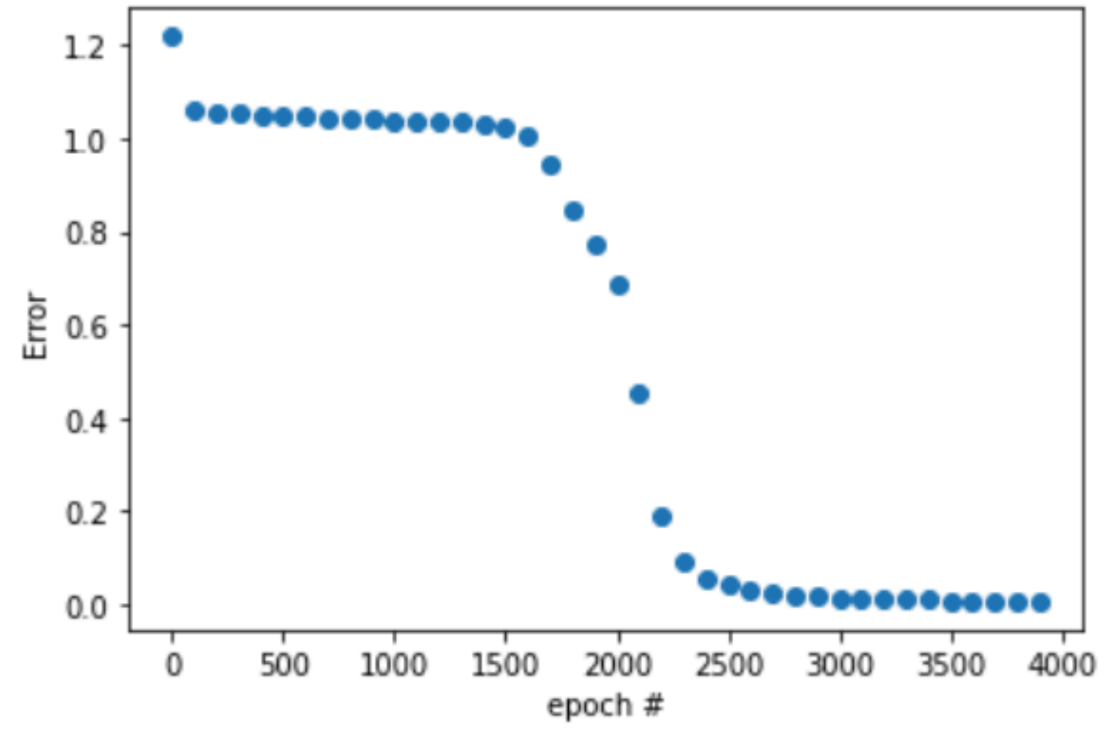


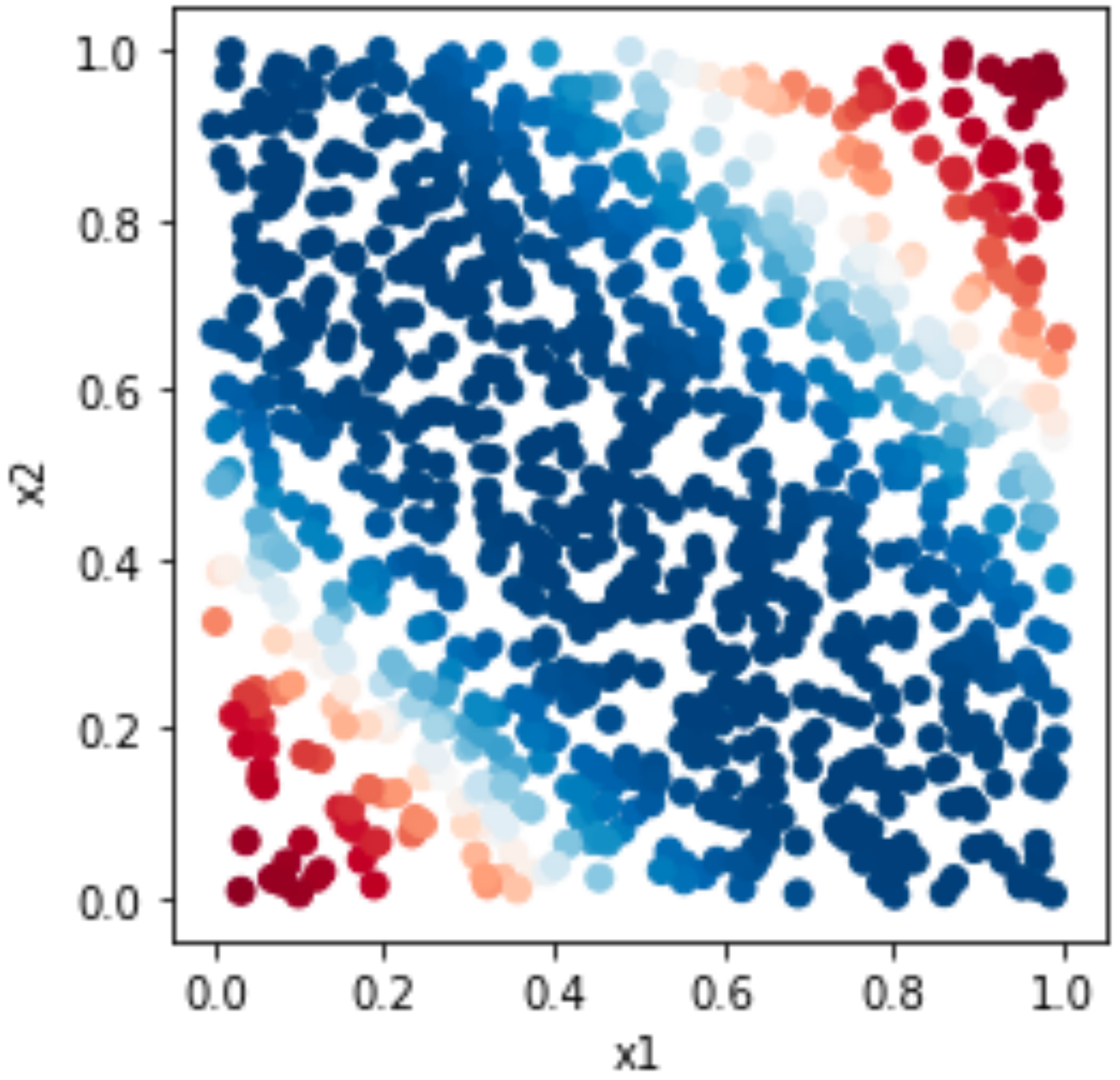
Example - XOR operator

```python
N = 4 # number of traning set
Nep = 4000 # number of epoch
alpha = 0.5 # learning rate

def h(x):
    return 1.0/(1.0 + np.exp(-x))

x_train = np.array( [[1,1], [1,0], [0,1], [0,0]] ) # training set
d_train = np.array( [0, 1, 1 ,0] ) # right answer

W = np.random.random( (2, 2) ) # random weights
b = np.random.random( (2, 1) )

V = np.random.random( (1, 2) ) # random weights
c = np.random.rand()

xarr = []; yarr = []

for ep in range(Nep):
    sume = 0
    for n in range(N):
        x = np.reshape(x_train[n], (2,1))
        d = d_train[n]

        v = W@x + b; y = h(v)
        s = V@y + c; z = h(s)
        e = d - z
        sume += np.ndarray.item(e*e)

        delta = z*(1-z)*e
        e1 = V.T @ delta
        epsil = y*(1-y)*e1

        V += alpha*delta*y.T; c += alpha*delta
        W += alpha*epsil*x.T; b += alpha*epsil

    if (ep % 100 == 0):
        xarr.append(ep); yarr.append(sume)
```




```
0 [1 1] 0 [[0.03600886]]
1 [1 0] 1 [[0.95303802]]
2 [0 1] 1 [[0.96323028]]
3 [0 0] 0 [[0.04028346]]
```



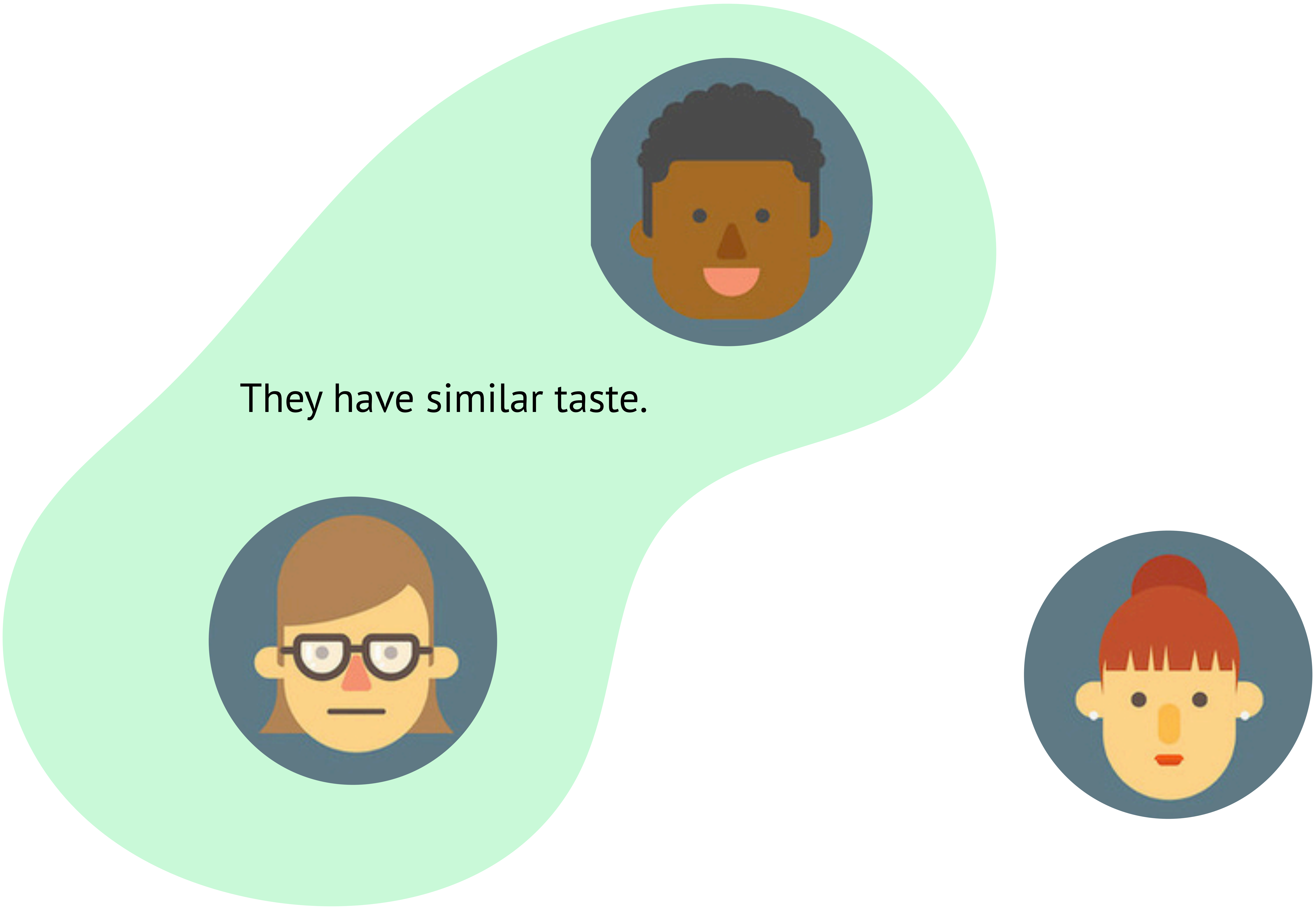


Did you listen to Chang Kiha's new songs?
It's amazing. You should listen to them.

Oh! Really? I will listen to them. All the songs
which you recommended were awesome.

Did you listen to Chang Kiha's new songs?
It's amazing. You should listen to them.

I don't buy it. To me, his song sounds weird.


They have similar taste.

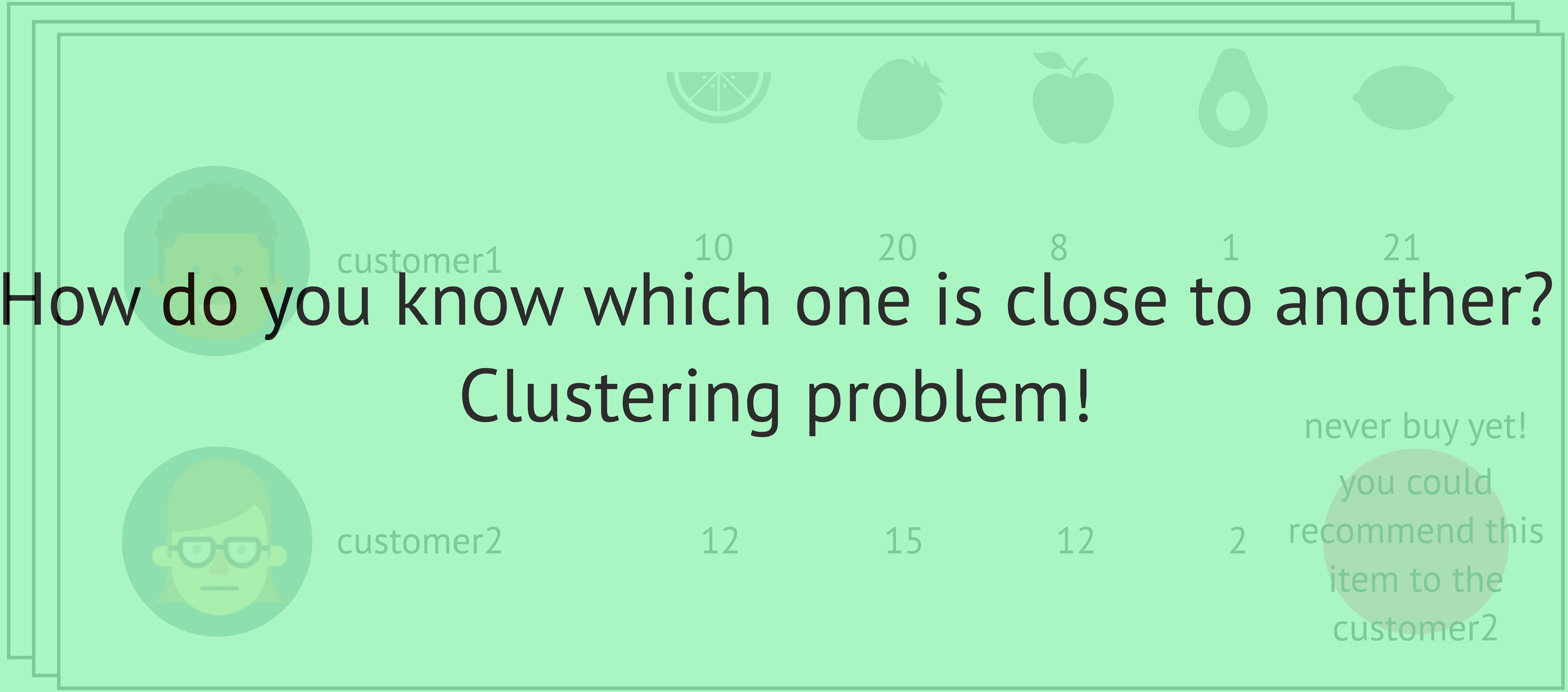
If you know two customers have similar taste, you could recommend some items based on their shopping patterns (which items the customers buy).

customer1	10	20	8	1	21
customer2	12	15	12	2	

If you know two customers have similar taste, you could recommend some items based on their shopping patterns (which items the customers buy).

If you know two customers have similar taste, you could recommend some items based on their shopping patterns (which items the customers buy).

If you know two customers have similar taste, you could recommend some items based on their shopping patterns (which items the customers buy).



How do you know which one is close to another?

Clustering problem!

Clustering

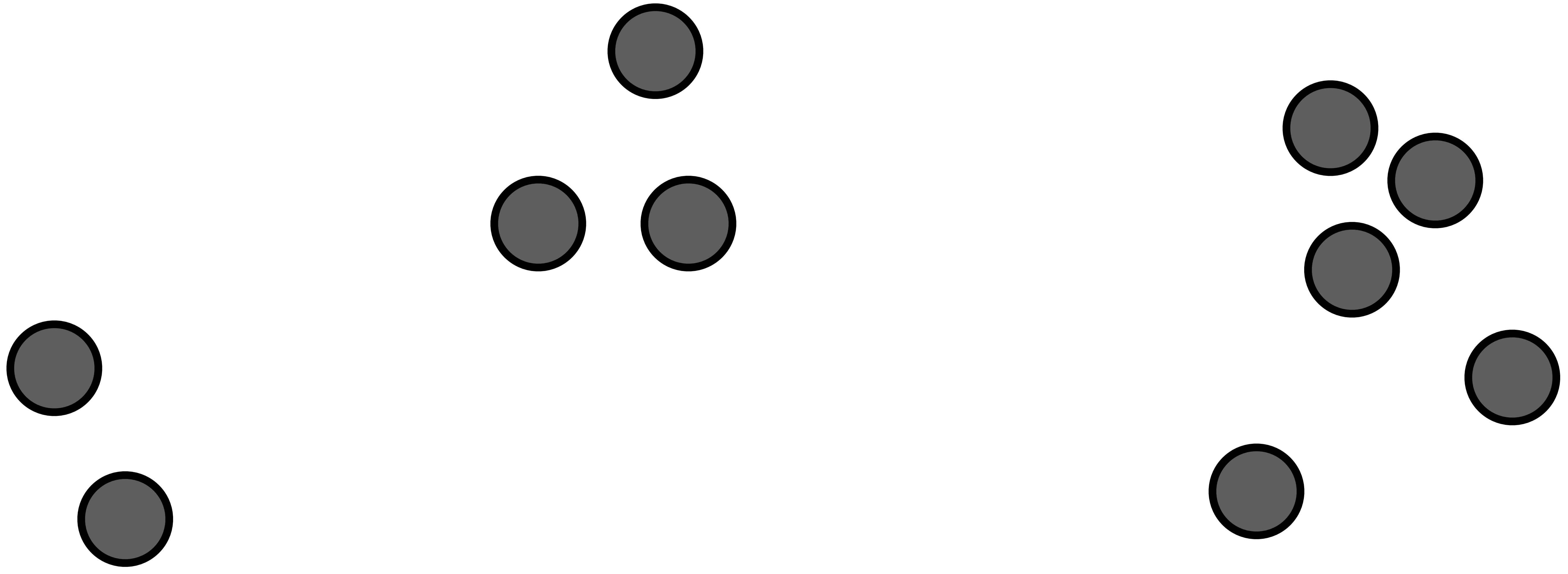
Cluster: Group of the similar objects

Good clustering?

⇨ High Intra-cluster similarity

⇨ Low Inter-cluster similarity

Example



Clustering

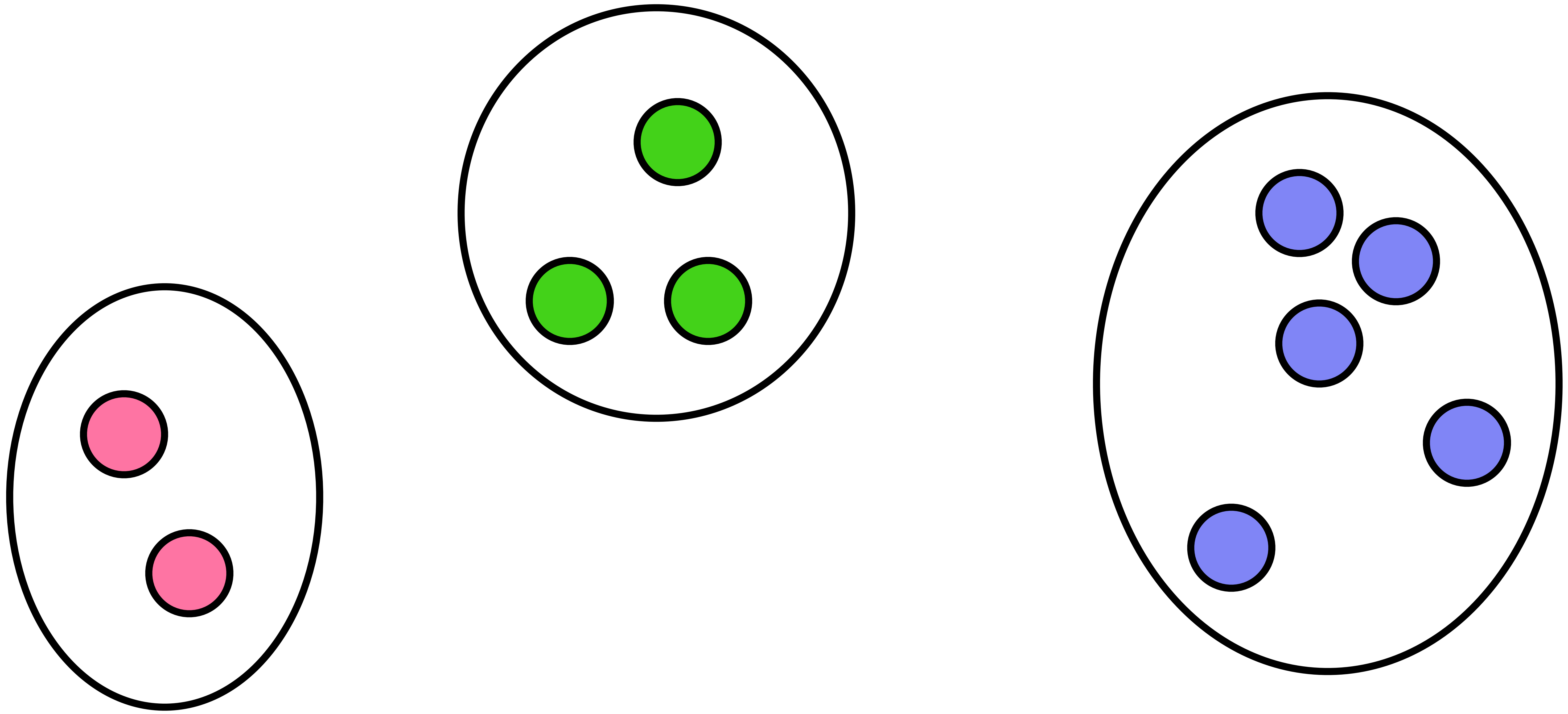
Cluster: Group of the similar objects

Good clustering?

⇨ High Intra-cluster similarity

⇨ Low Inter-cluster similarity

Example



Clustering

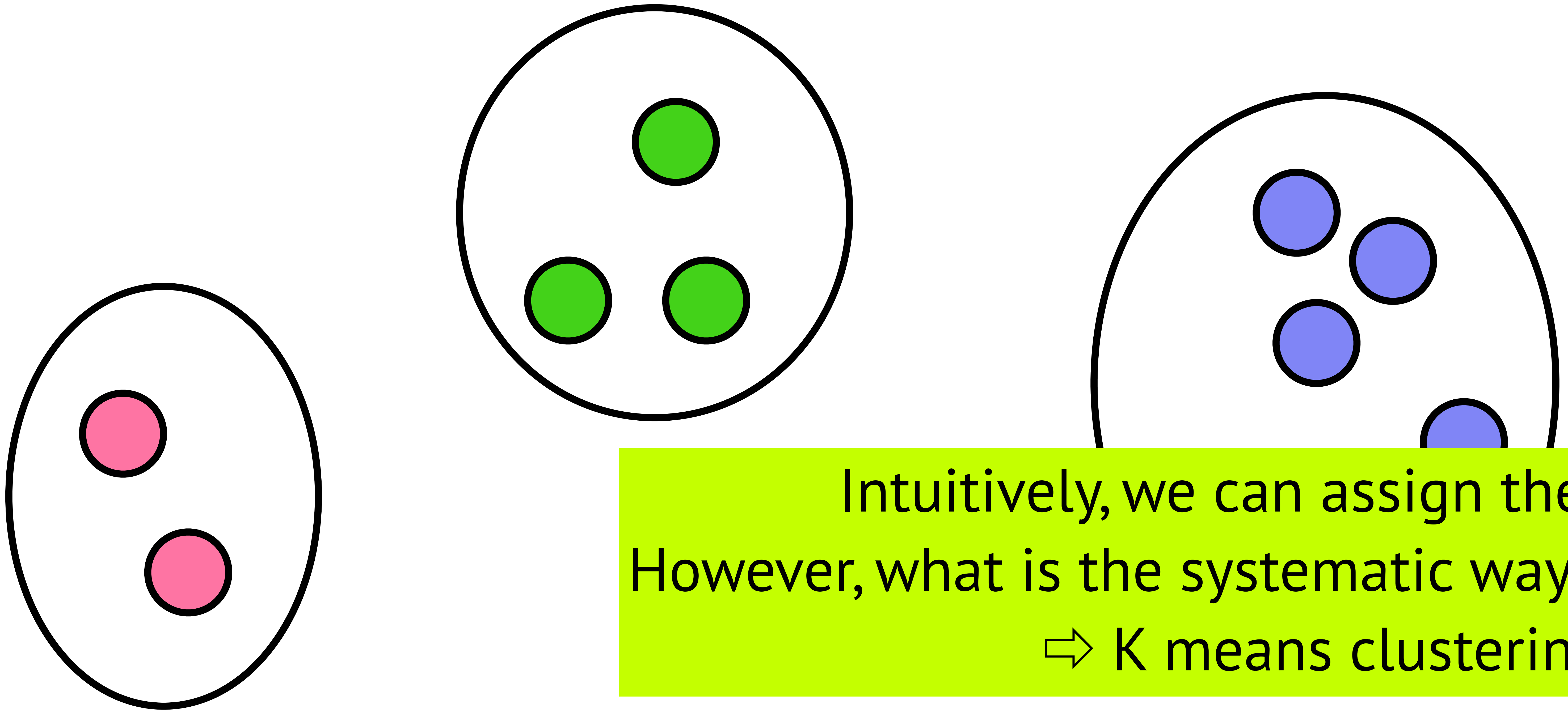
Cluster: Group of the similar objects

Good clustering?

⇨ High Intra-cluster similarity

⇨ Low Inter-cluster similarity

Example



Intuitively, we can assign the clusters.
However, what is the systematic way to do clustering?
⇨ K means clustering

K means clustering

We will use Euclidean distance to mention something is similar or not.
If two points are close, they are similar.

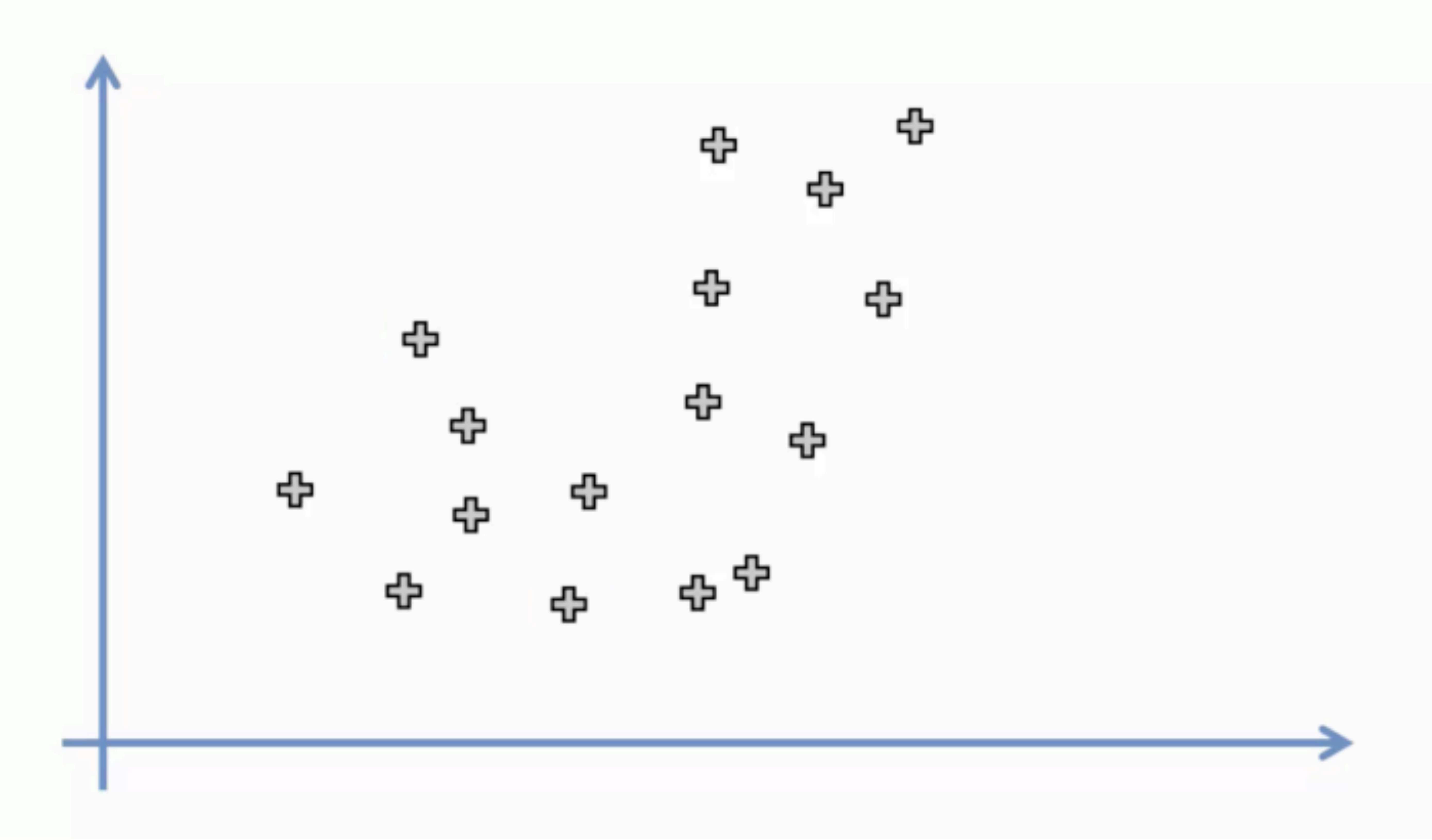
Algorithm

1. Choose K the number of clusters.
2. Randomly choose K samples (data points) for the initial cluster centroids.
3. For all data points, assign the cluster based on the distance between data point and the cluster centroid; the point belongs to the closest cluster centroid.
4. Recalculate the cluster centered based on allocation.
5. Repeat 3-4 until the clusters do not change.

K means clustering

https://towardsdatascience.com/machine-learning-algorithms-part-9-k-means-example-in-python-f2ad05ed5203

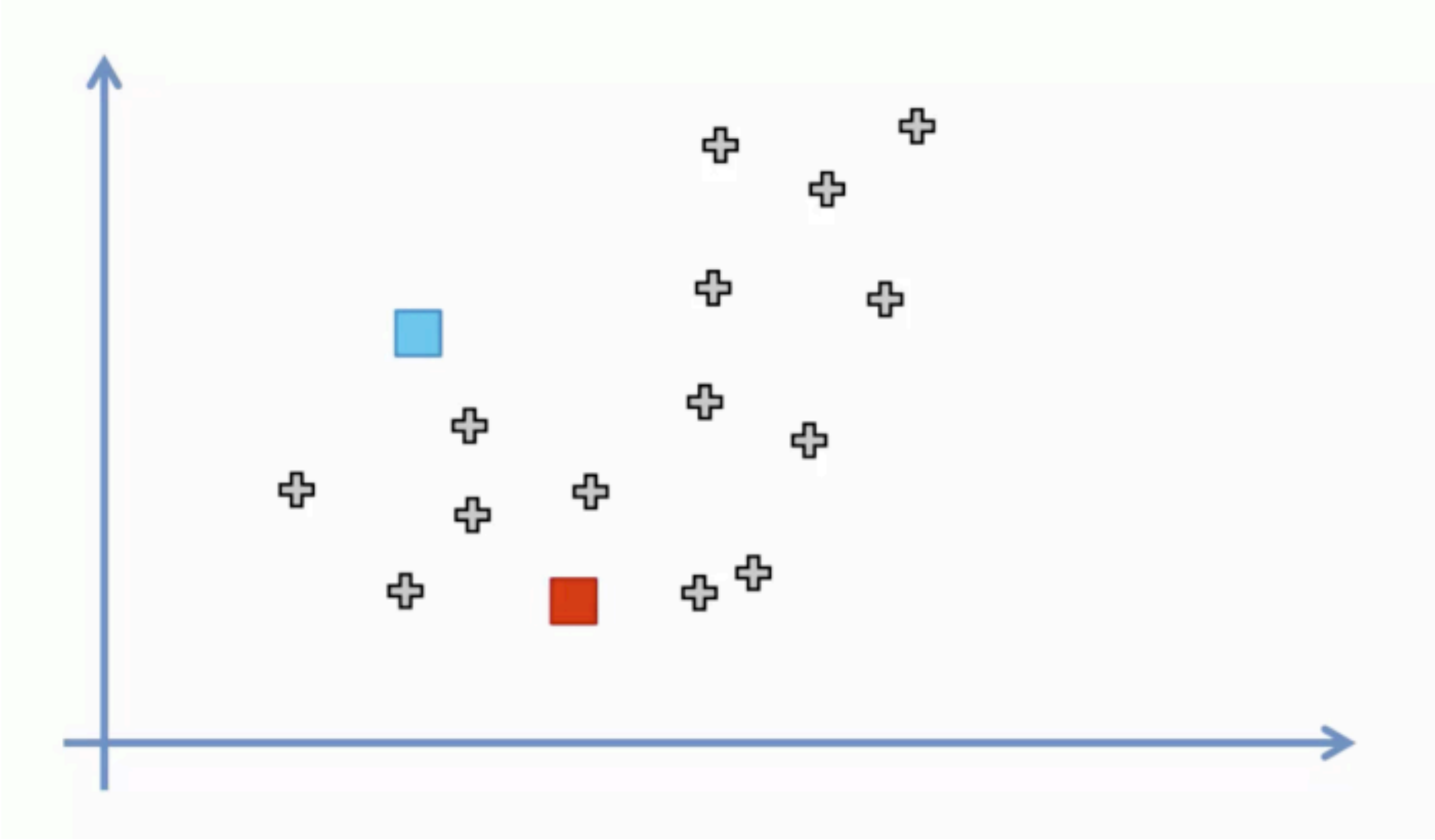
1. Choose K the number of clusters (let's say 2 in this example).



K means clustering

https://towardsdatascience.com/machine-learning-algorithms-part-9-k-means-example-in-python-f2ad05ed5203

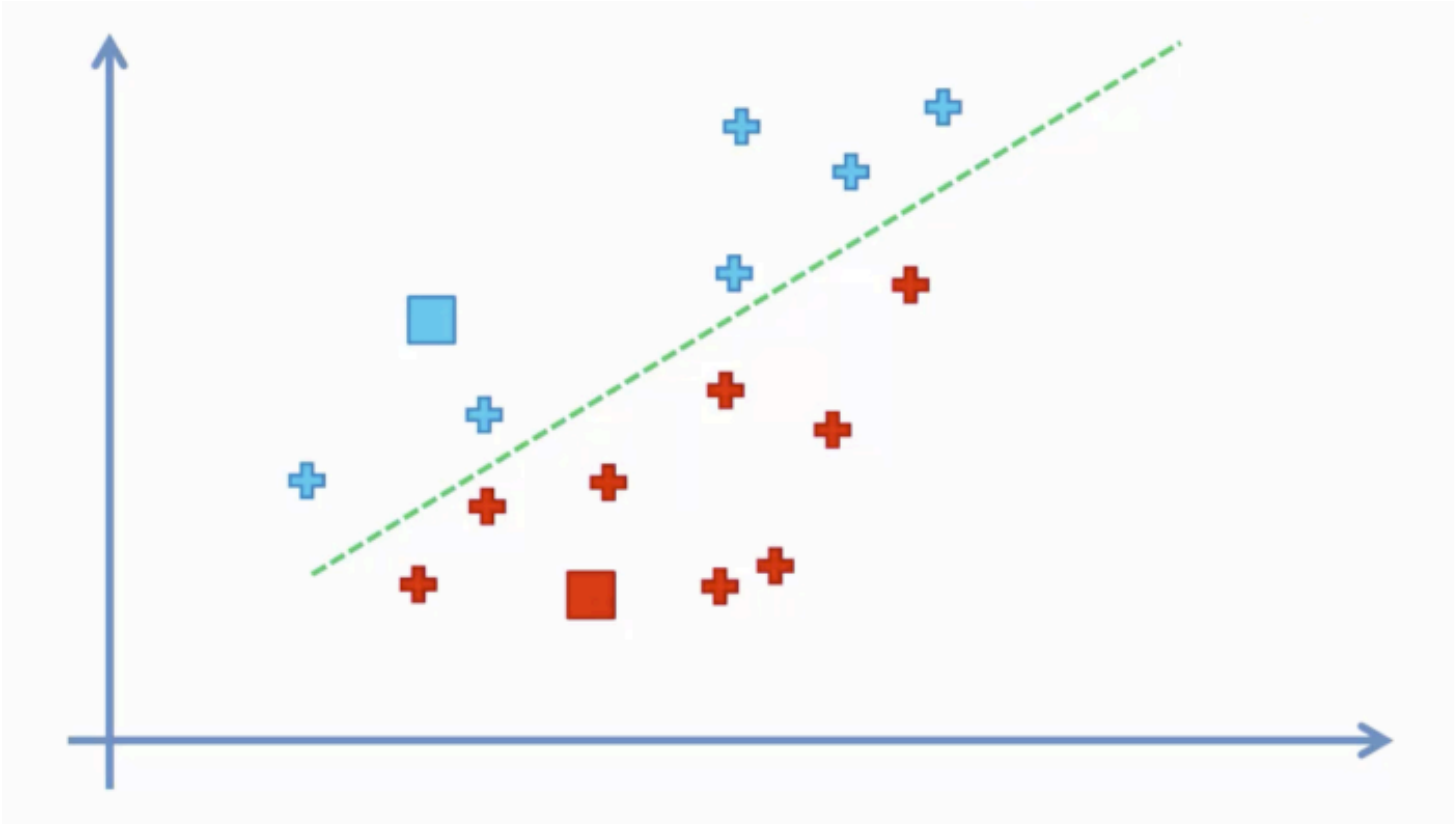
2. Randomly choose K samples (data points) for the initial cluster centroids.



K means clustering

https://towardsdatascience.com/machine-learning-algorithms-part-9-k-means-example-in-python-f2ad05ed5203

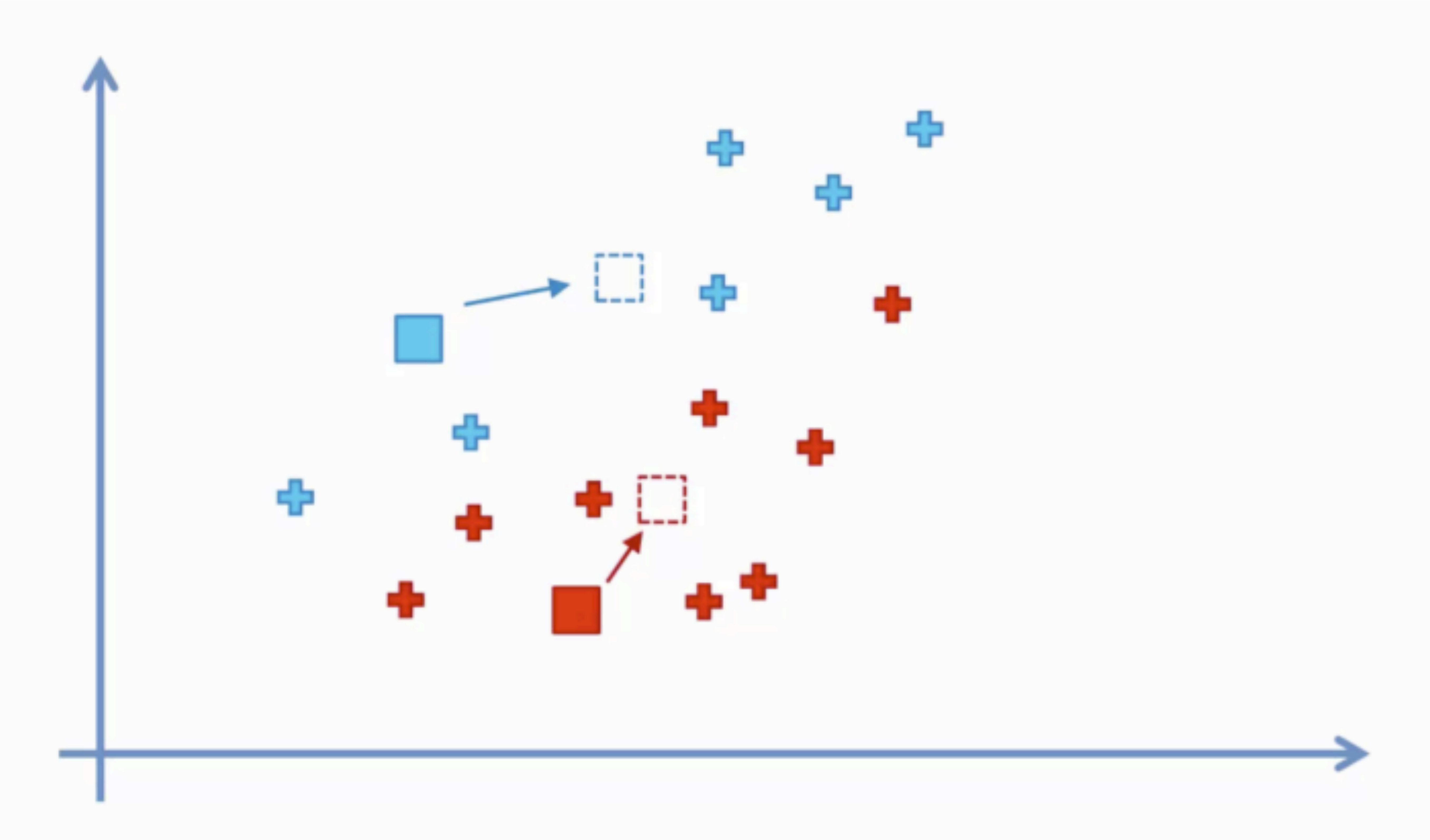
3. For all data points, assign the cluster based on the distance between data point and the cluster centroid; the point belongs to the closest cluster centroid.



K means clustering

https://towardsdatascience.com/machine-learning-algorithms-part-9-k-means-example-in-python-f2ad05ed5203

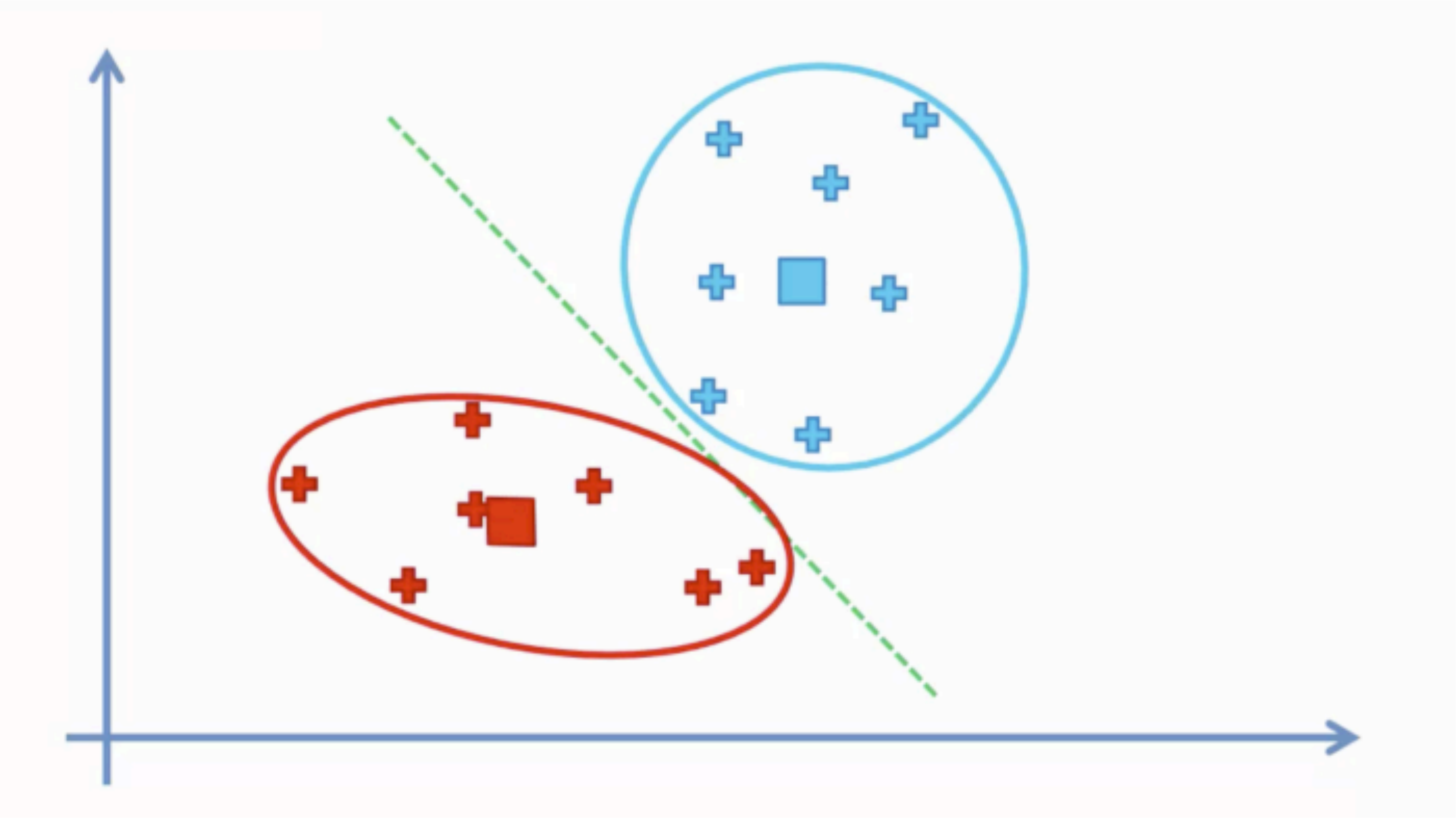
4. Recalculate the cluster centered based on allocation.



K means clustering

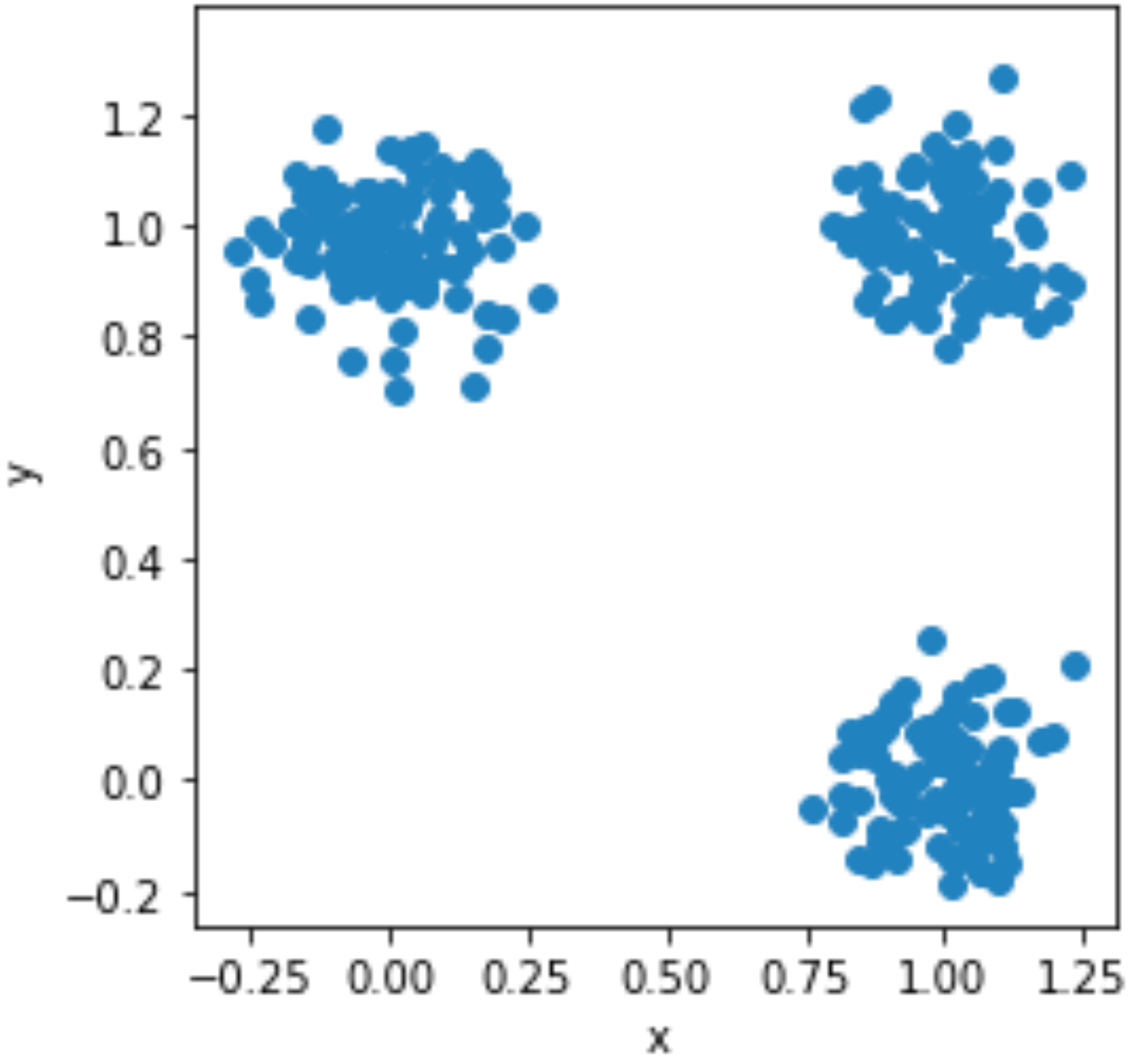
https://towardsdatascience.com/machine-learning-algorithms-part-9-k-means-example-in-python-f2ad05ed5203

5. Repeat 3-4 until the clusters do not change.

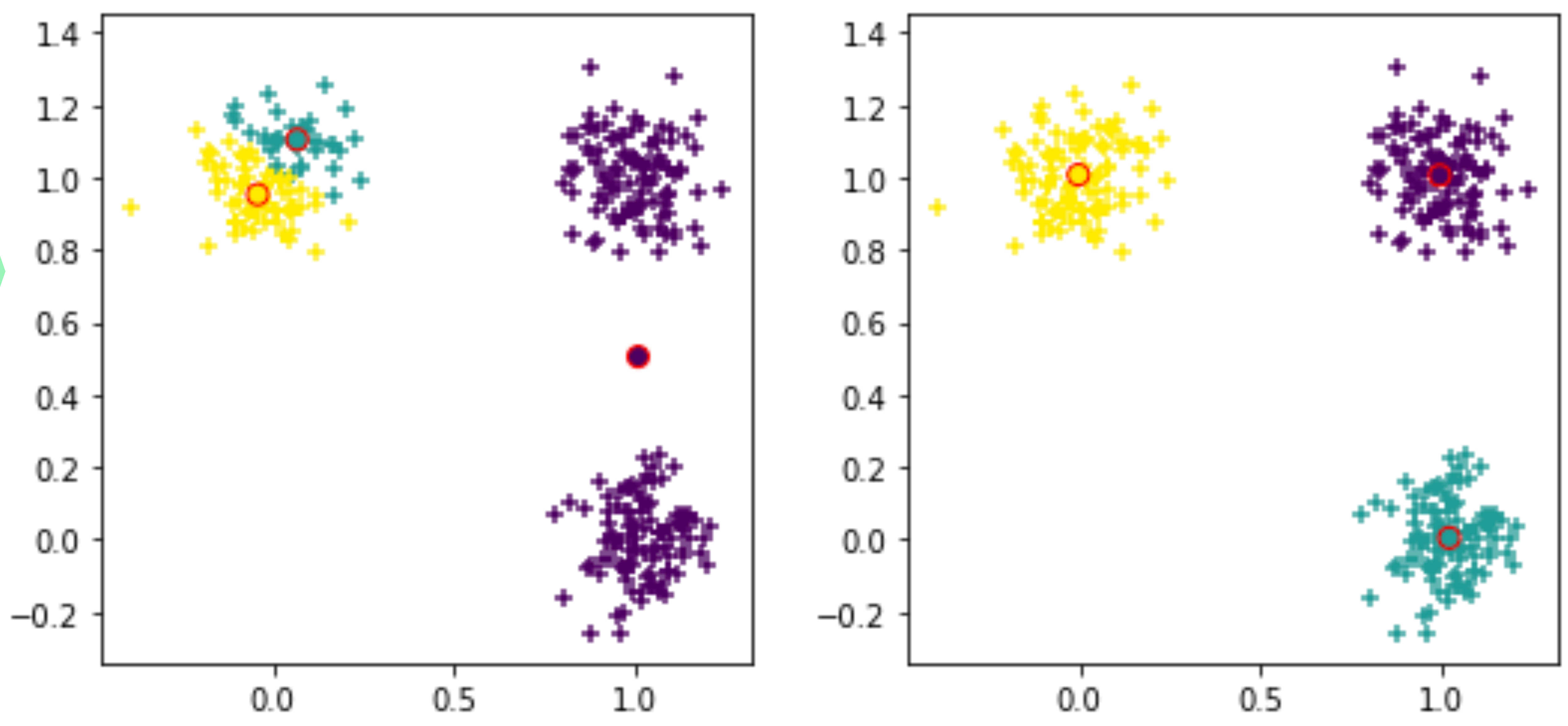


Initial condition dependency

A randomly generated data set



K means clustering results strongly depends on initial centroids.



Kmeans++

[https://itstory1592.tistory.com/19]

The way of selecting an initial centroid set

When the randomly selected initial centroids are close each other, K means clustering usually unstable. To overcome this issue, we could use Keans++ method that chooses initial centroids as far as possible.

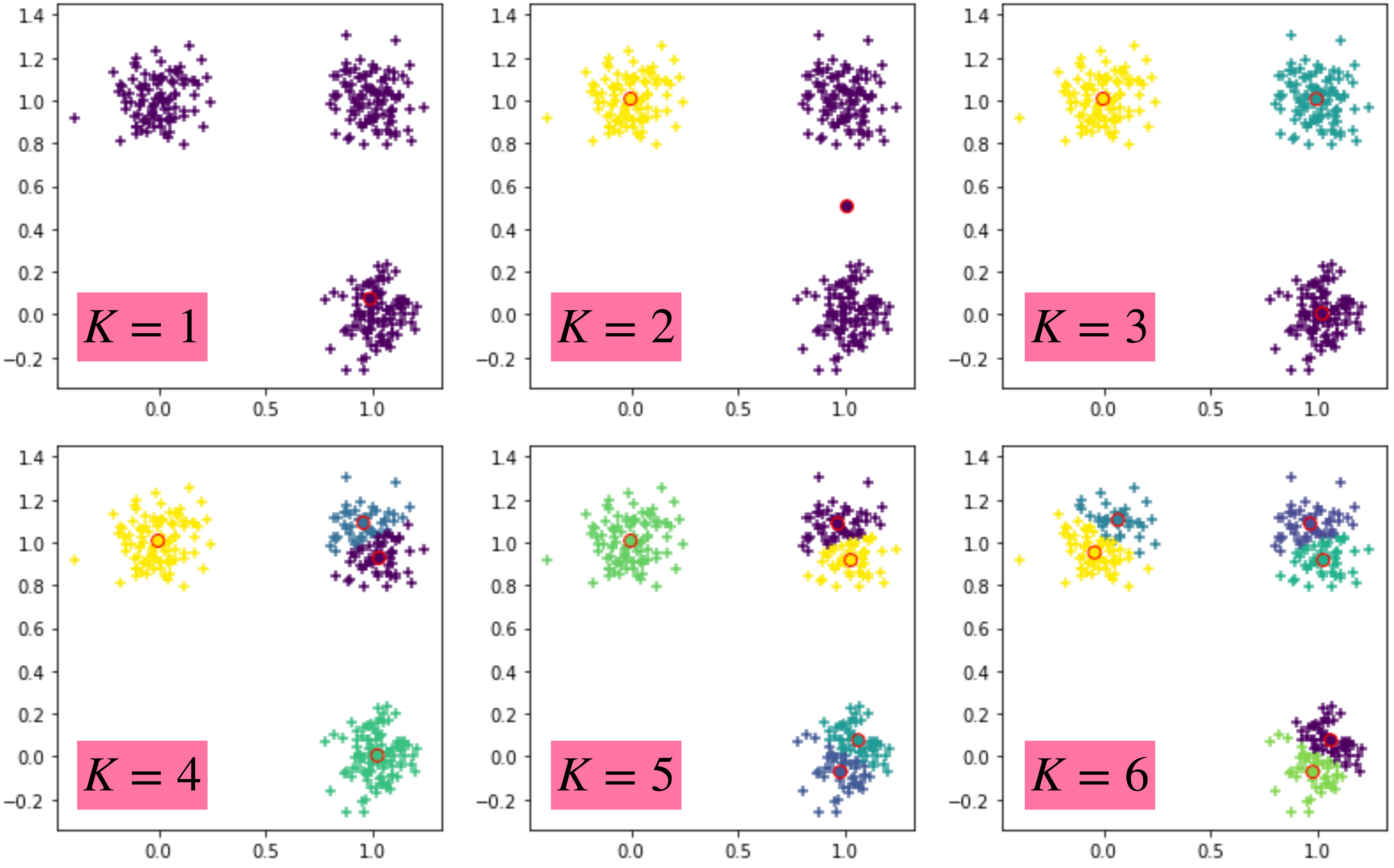
Algorithm

1. Choose the first centroid at random.
2. Calculate the distances from points to the centroid.
3. Depending on distance, choose the next centroid as

$$p_i = \frac{\text{distance between } i\text{-th data point and the centroid, } d_i}{\sum_i d_i}$$

4. Repeat 2-3 until K centroids are selected.

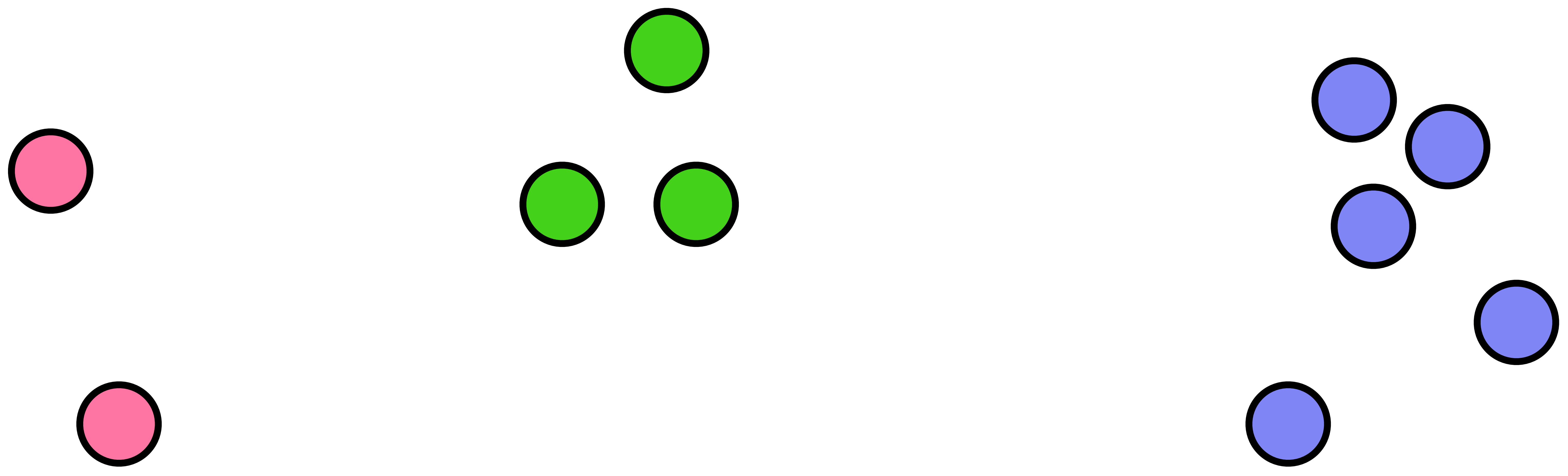
How to decide K



How to decide K

Elbow method

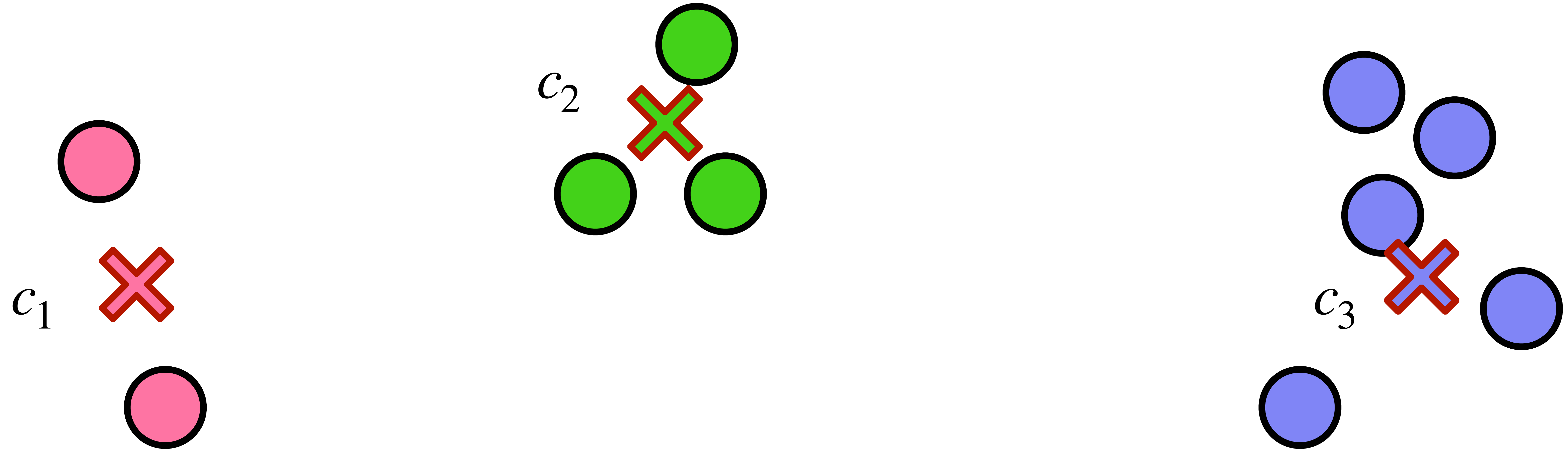
Minimizing the error within clusters, $E = \sum_{j=1}^{K} \sum_{i \in c_j} d(x_i, c_j)^2$



How to decide K

Elbow method

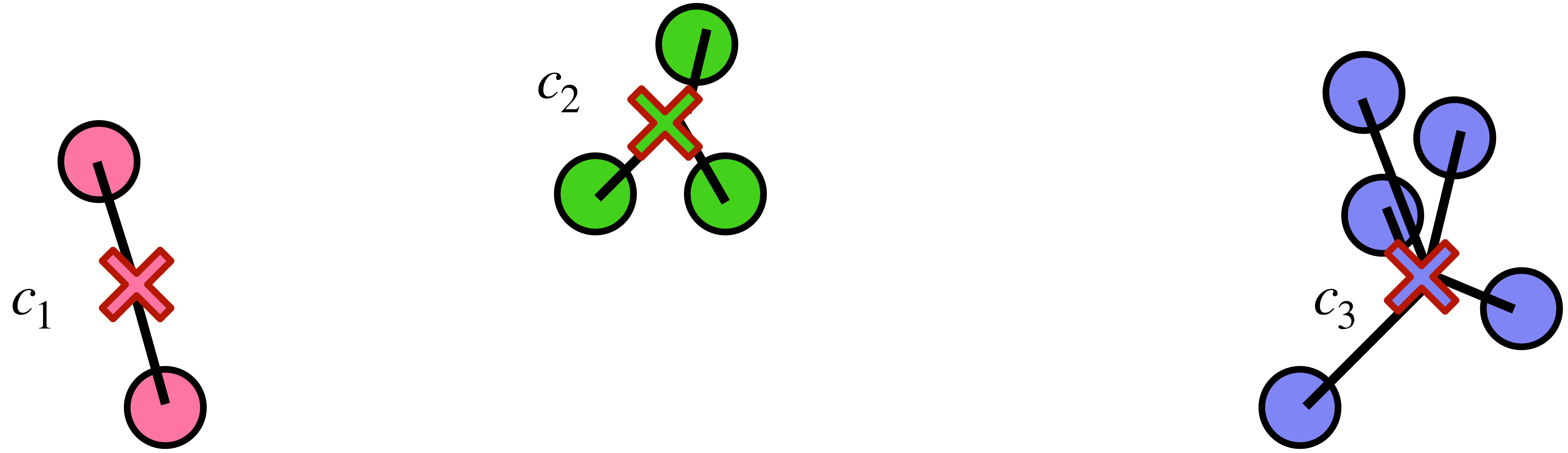
Minimizing the error within clusters, $E = \sum_{j=1}^{K} \sum_{i \in c_j} d(x_i, c_j)^2$



How to decide K

Elbow method

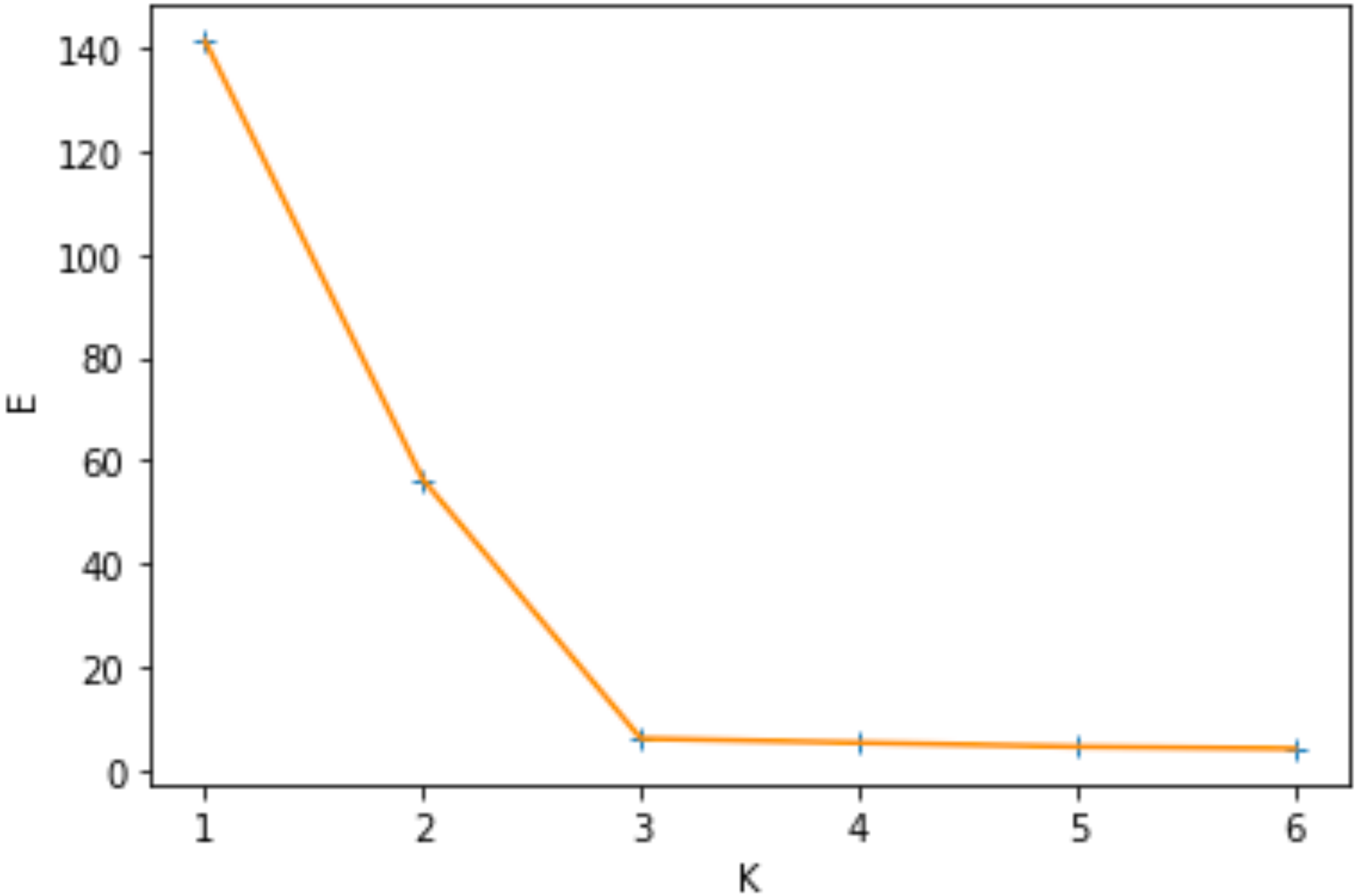
Minimizing the error within clusters, $E = \sum_{j=1}^{K} \sum_{i \in c_j} d(x_i, c_j)^2$



How to decide K

Elbow method

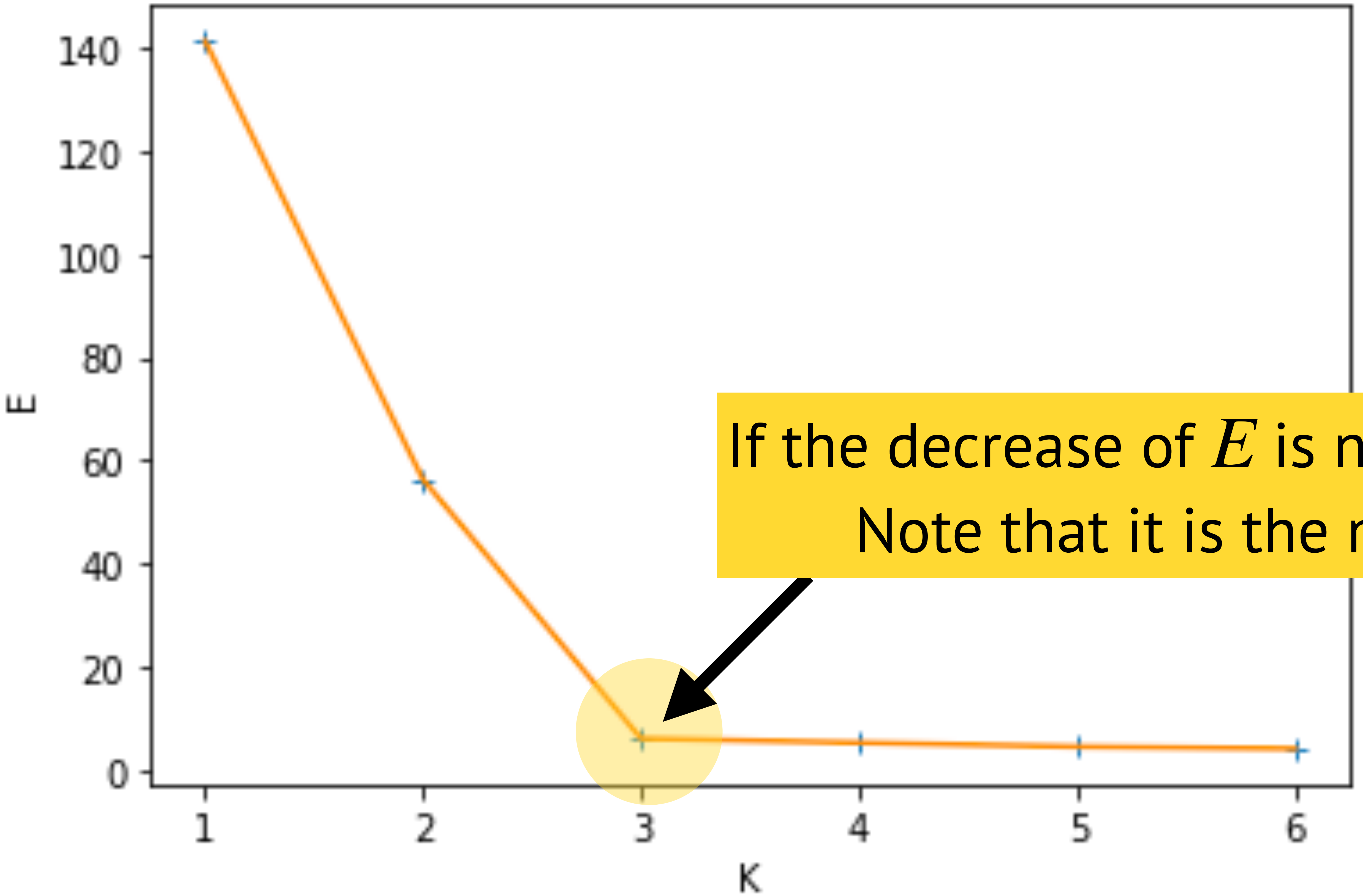
Minimizing the error within clusters, $E = \sum_{j=1}^{K} \sum_{i \in c_j} d(x_i, c_j)^2$



How to decide K

Elbow method

Minimizing the error within clusters, $E = \sum_{j=1}^{K} \sum_{i \in c_j} d(x_i, c_j)^2$



If the decrease of E is not significant, we can stop to increase K.
Note that it is the minimum point but still meaningful.

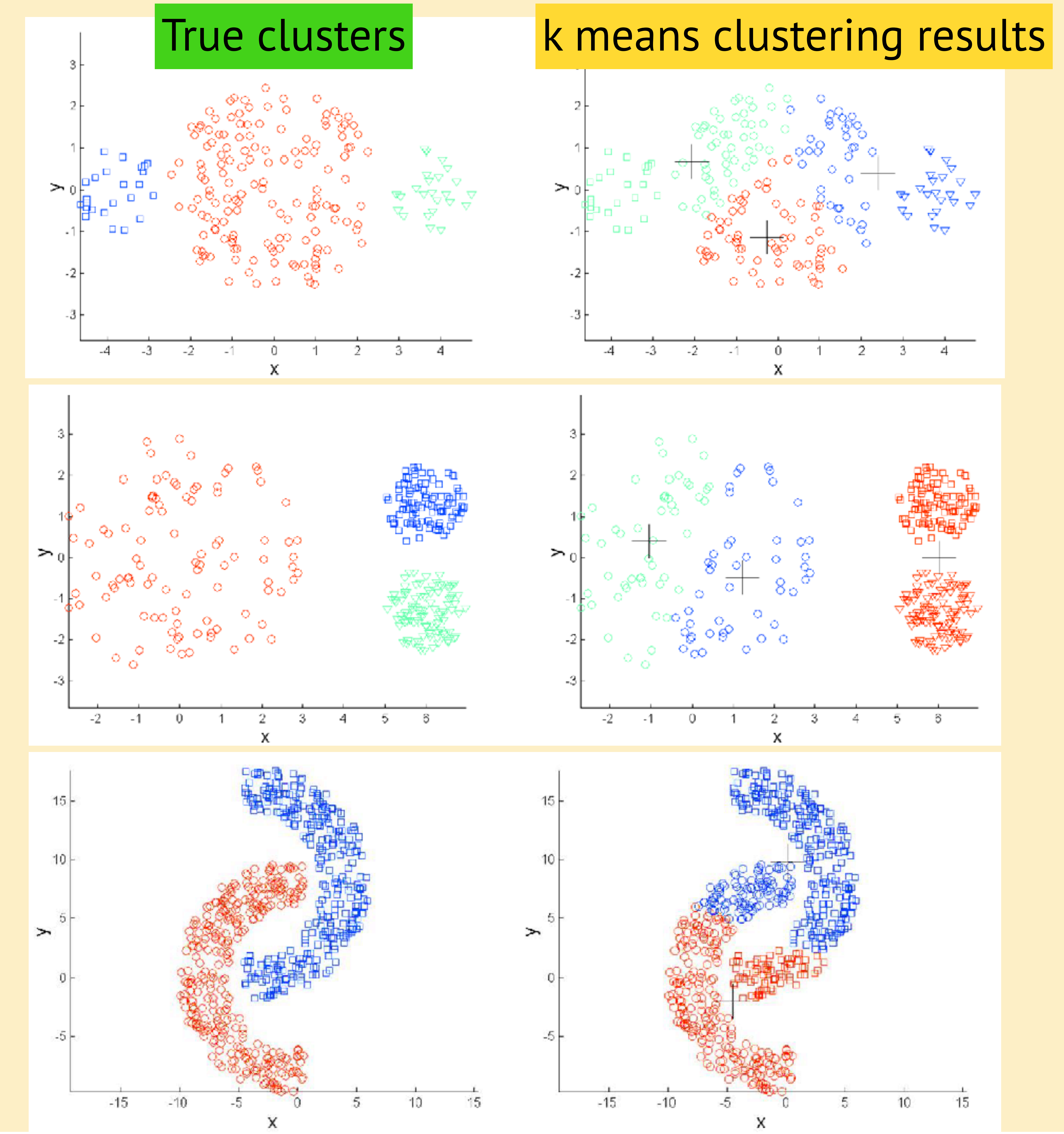
Limitation

[https://ratsgo.github.io/machine%20learning/2017/04/19/KC/]

- Different data volume for each cluster

- Different data density for each cluster

- Nonconvex shapes of the data

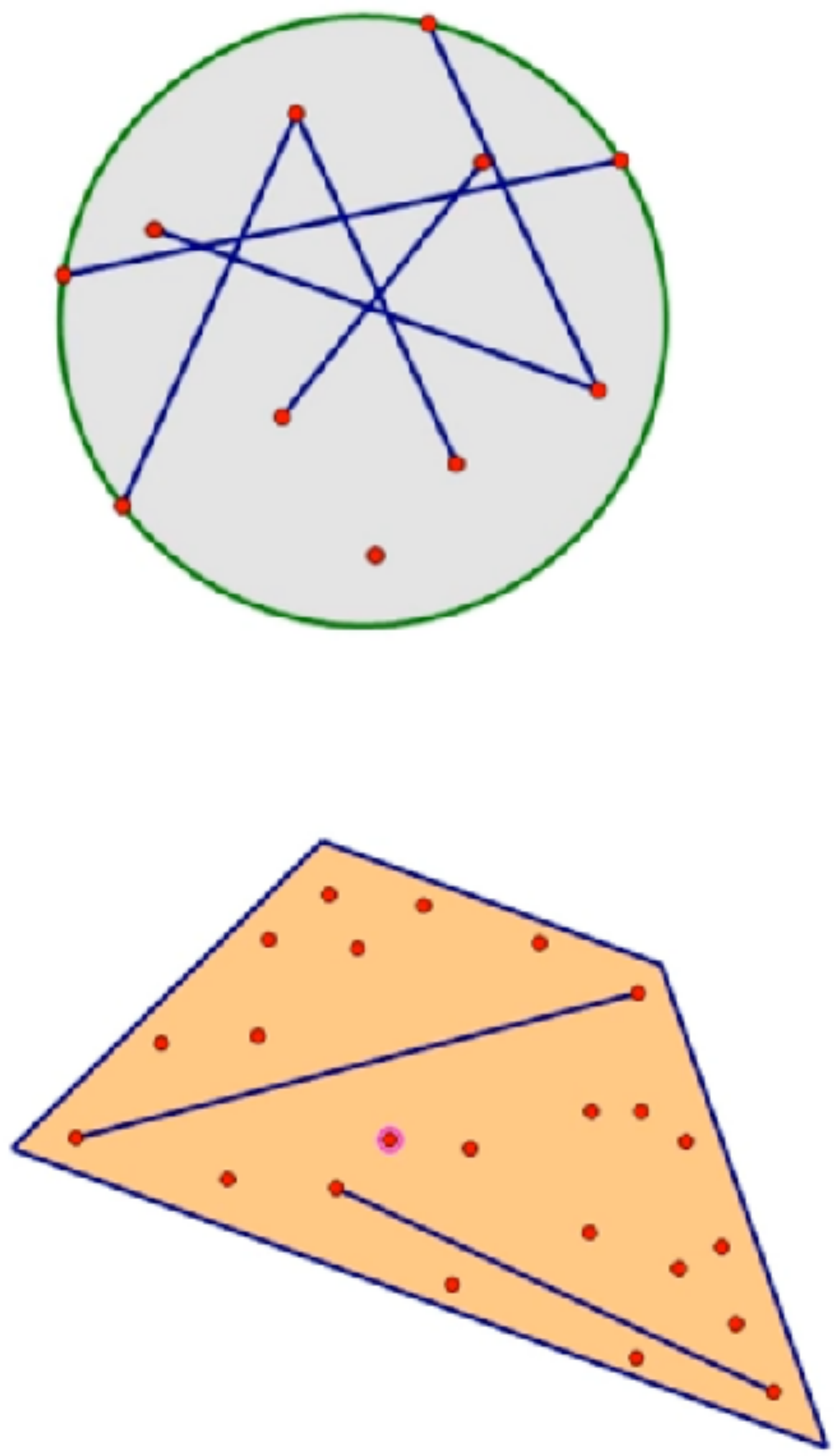


[cf] Convex and Nonconvex

https://www.youtube.com/watch?v=hrSdndm4EPA

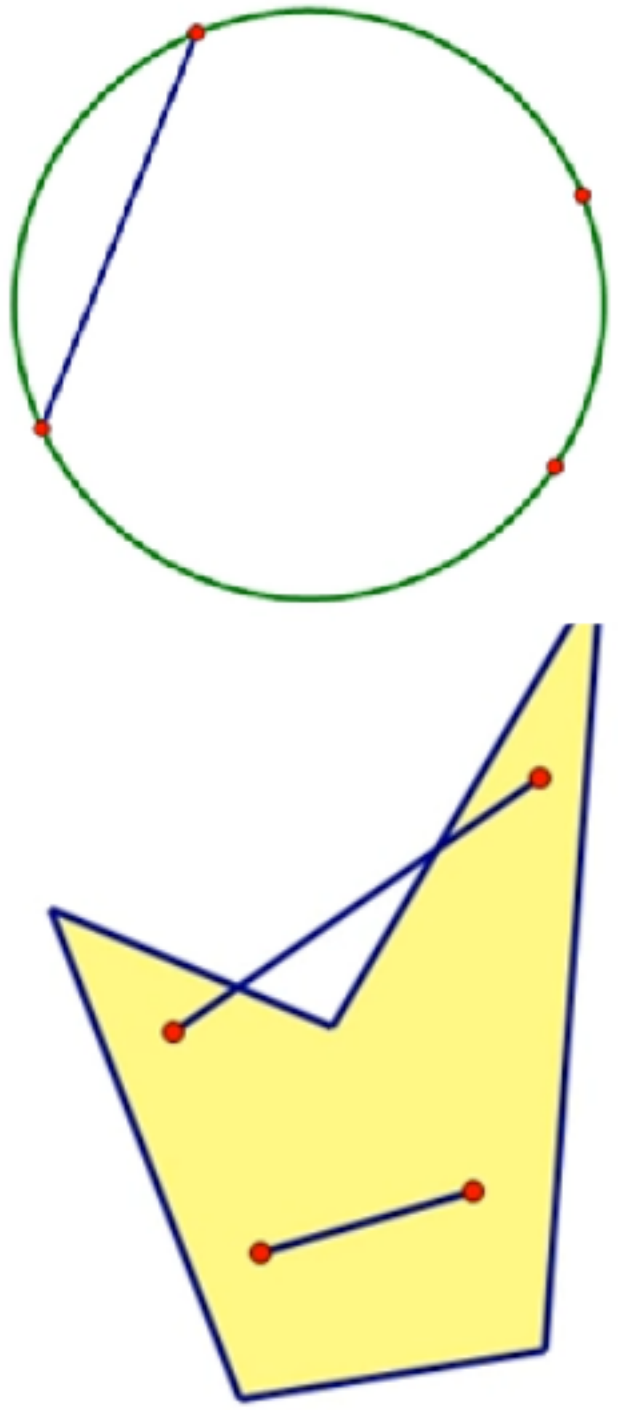
Convex

any lines connecting two points are in a set.



Nonconvex

there is a line (or lines) connecting two points are not in a set.



Code

```
#array for cluster idx of each data point
cluster = np.zeros(Npoints)

#step1: choose the number of clusters
K = 3

#step2: randomly select K samples among data
centroids = data[np.random.randint(Npoints, size=K)]

flag = 1
while(flag):
  #step3: calculate the distance between and alocate the cluster idx for data points
  ex = cluster.copy()
  for i in range(Npoints):
    dist = distances(data[i], K, centroids) #calculate distance
    idx = np.argmin(dist) #find the minimum
    cluster[i] = idx #allocate the cluster

  #step4: recalculate the centroids
  for i in range(K):
    centroids[i] = data[np.where(cluster==i)].mean(axis=0)

  #step5: termination condition
  if np.array_equal(ex, cluster):
    flag = 0
```

References

- https://www.youtube.com/watch?v=4b5d3muPQmA
- https://todayisbetterthanyesterday.tistory.com/58
- https://towardsdatascience.com/machine-learning-algorithms-part-9-k-means-example-in-python-f2ad05ed5203
- https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-k-means-clustering/