Ứng dụng Trí tuệ nhân tạo trong Nuôi trồng thủy sản

NGUYỄN HẢI TRIỀU[1]

[1] Bộ môn Kỹ thuật phần mềm,

Khoa Công nghệ thông tin, Trường ĐH Nha Trang

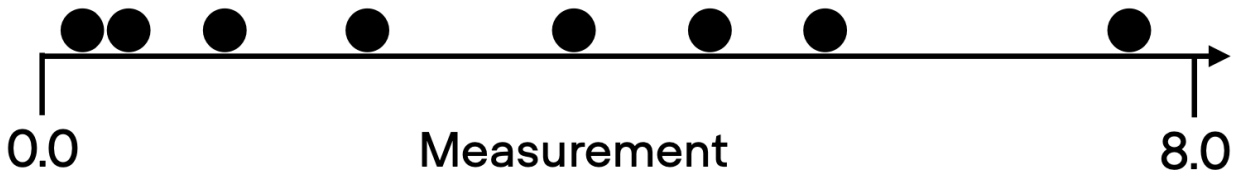
NhaTrang, September 2024

Trường hợp đơn giản

A dataset with only 1 feature variable (measurement). The goal here is to classify which class these points belong to. Also to keep things simple, we will start with binary classification.

Inputs **Labels**

Measurement	
1.4	0
1.1	0
5.1	1
6.2	1
9.9	1
2.2	0
3.3	0
7.2	1

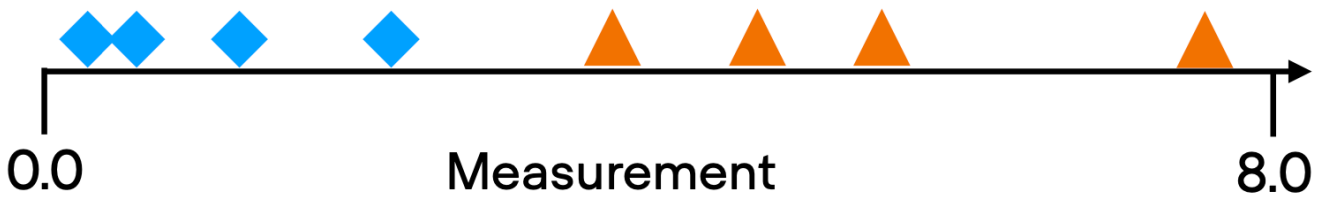


Consider the simplest case – Examples of Binary Classification:

- Spam/not-Spam
- Cat or dog
- Loan approved or not

Inputs Labels

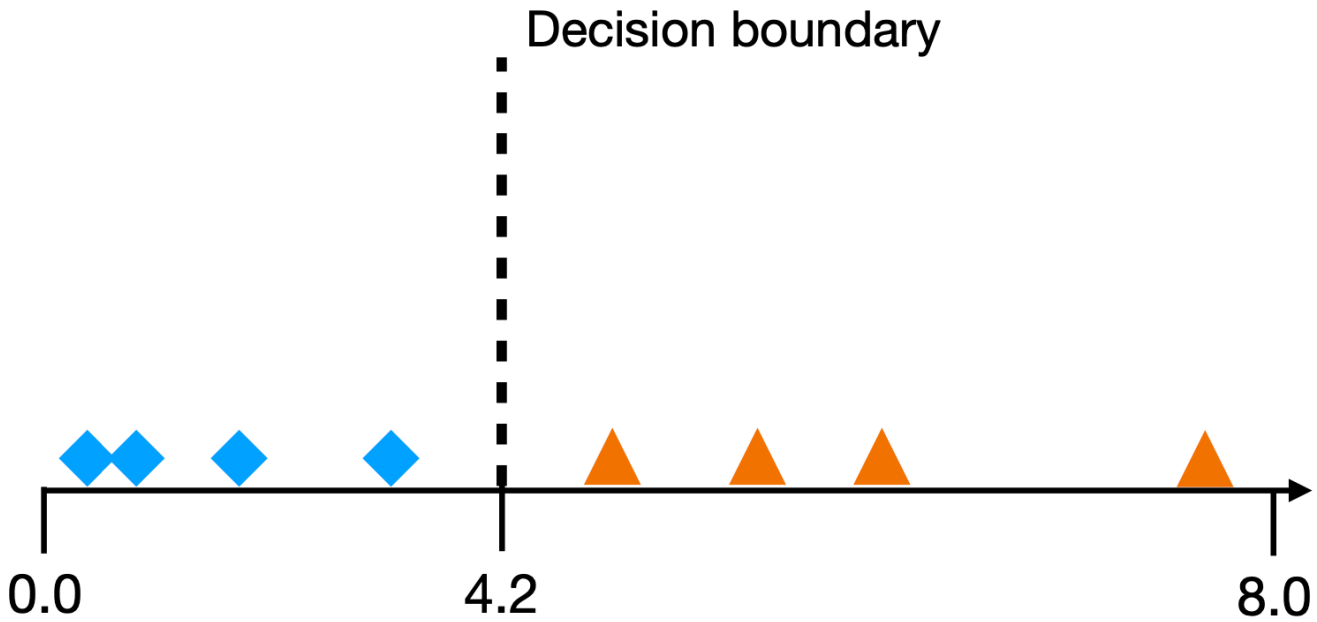
Measurement	
1.4	0
1.1	0
5.1	1
6.2	1
9.9	1
2.2	0
3.3	0
7.2	1



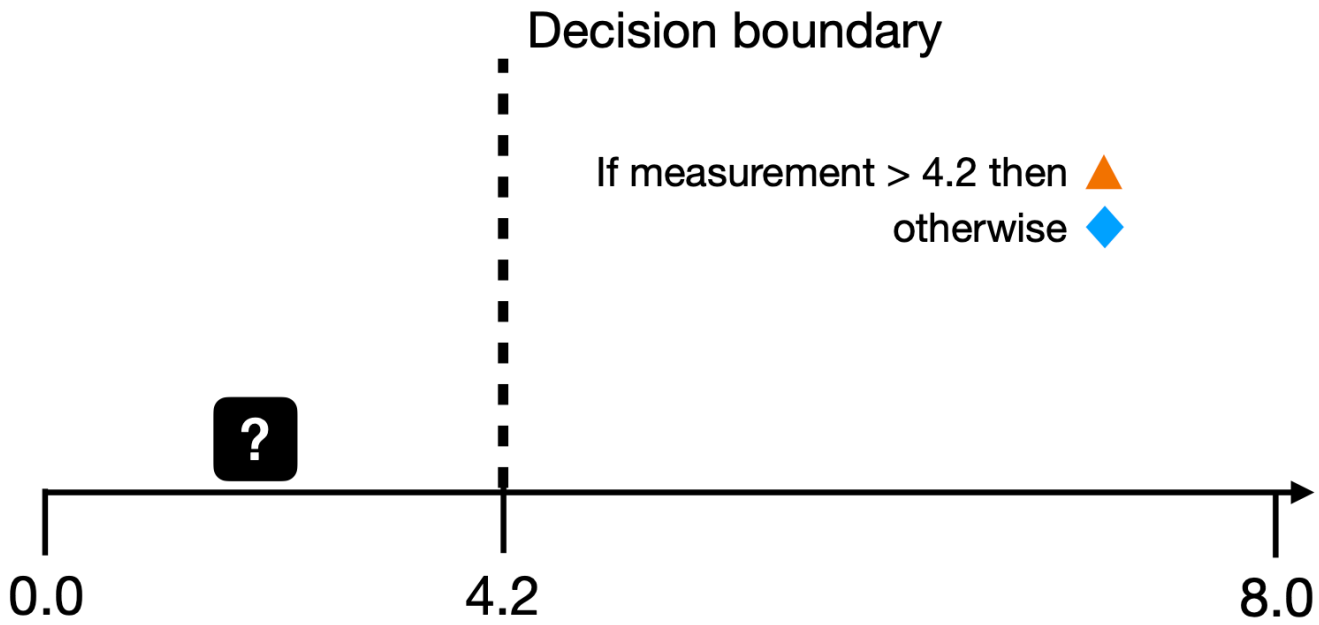
Now that we have a labeled data set, the goal is to predict whether a new data point belongs to the blue diamond or the orange triangle class.

Inputs Labels

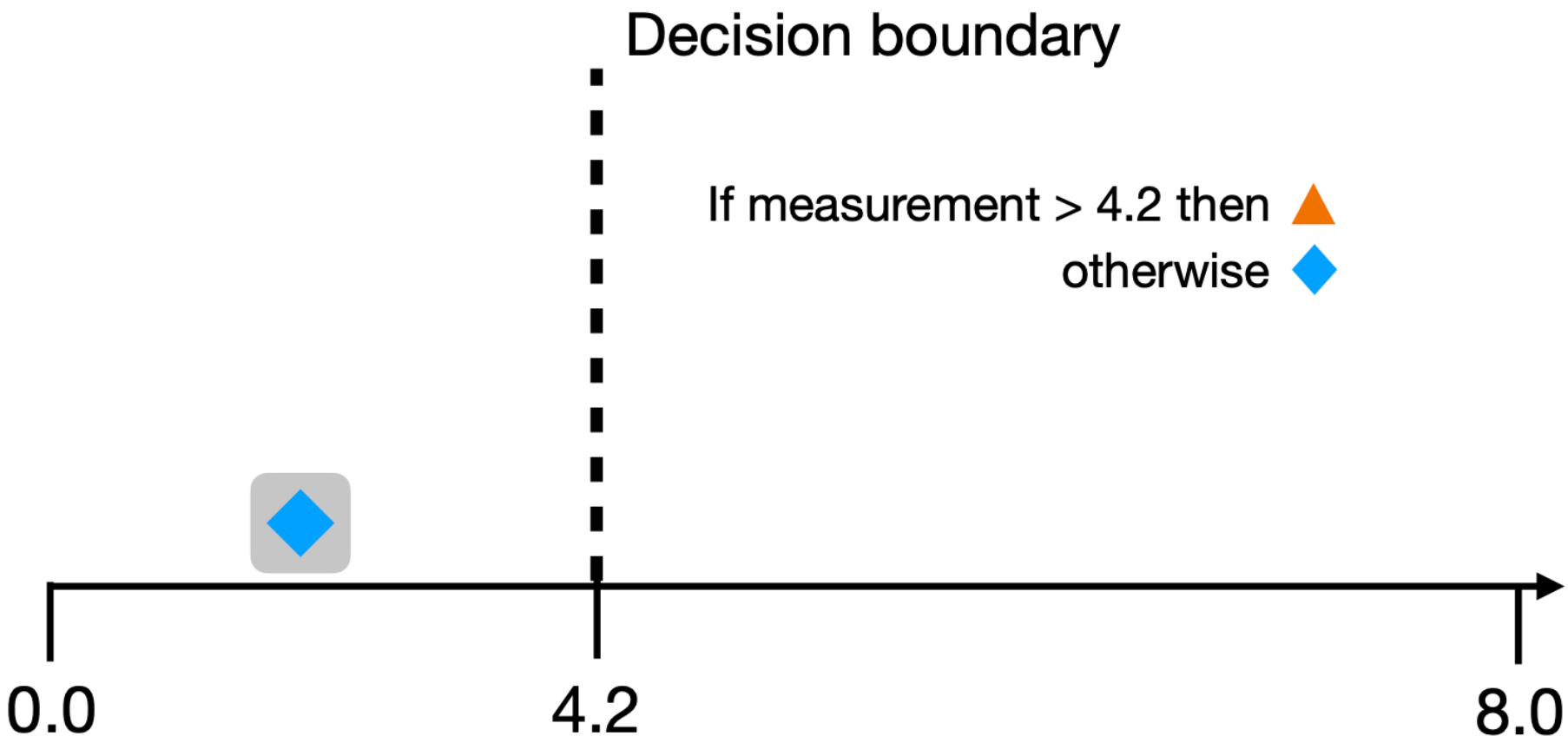
Measurement	
1.4	0
1.1	0
5.1	1
6.2	1
9.9	1
2.2	0
3.3	0
7.2	1



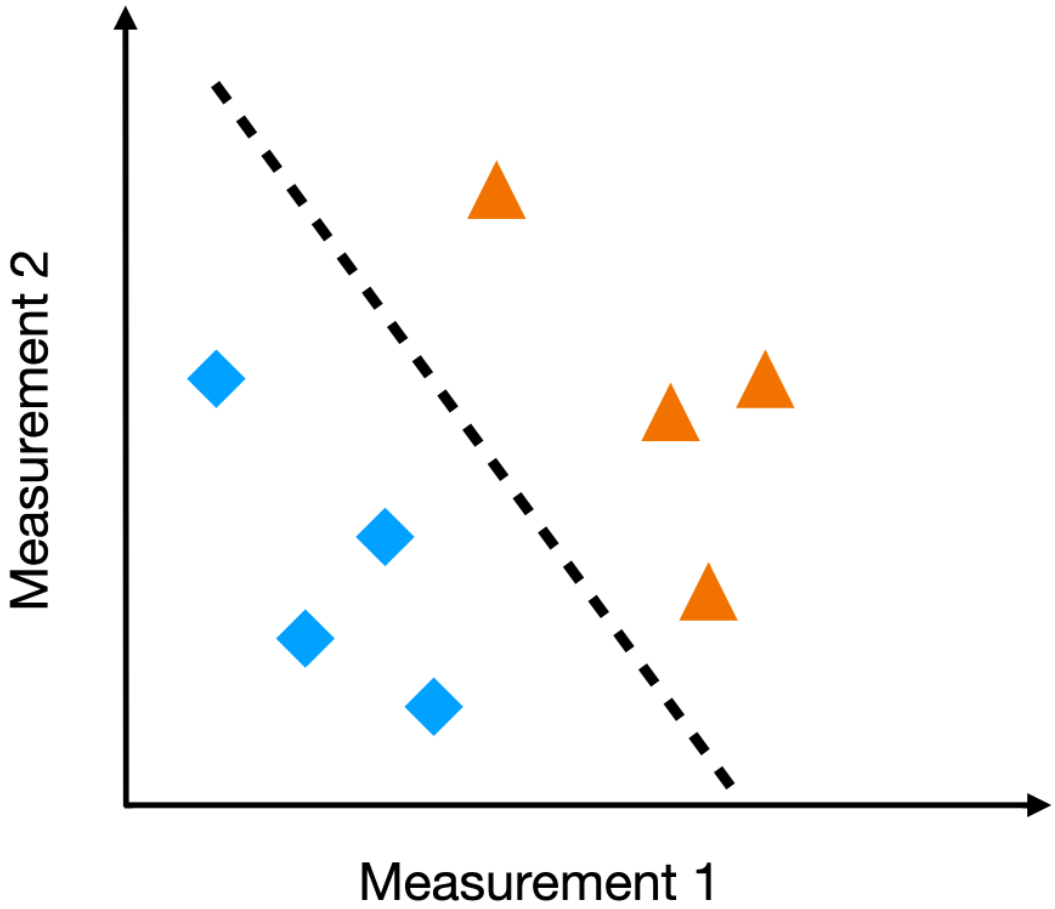
Decision boundary: we can say any new point that falls on the left side of the decision boundary is a blue diamond. And any point that falls on the right side of this decision boundary is an orange triangle.



So if we are given a new data point, such as this question mark here placed somewhere between 0 and 4.2, we can use our decision boundary to classify it.

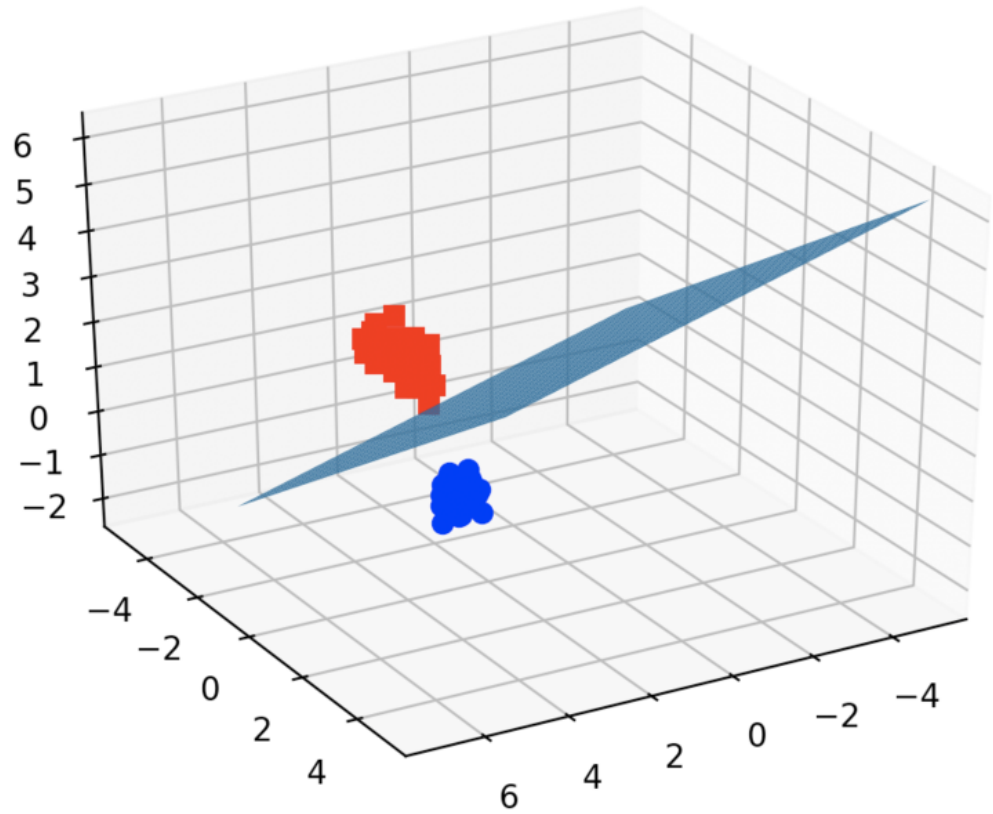

Decision boundary
If measurement > 4.2 then
otherwise
0.0
4.2
8.0

So real world problems often have more than one measurement or feature variable. So here's an example of a two dimensional data set where we have two measurement axes.



we can also draw a decision boundary, which involves both features

In case, we have three features, we would draw a two dimensional plane to separate these points. And we are looking here at a three dimensional data set.



Higher dimensions: Image classification problem

If you think about an image classification problem, each pixel in the image would be one feature. If we have, for example, a thousand pixel image, this **would be a thousand features**. We can't really visualize this in a plot anymore to draw this decision boundary, however, machine learning and deep learning systems are capable of plotting these decision boundaries in a higher dimensional space that goes.

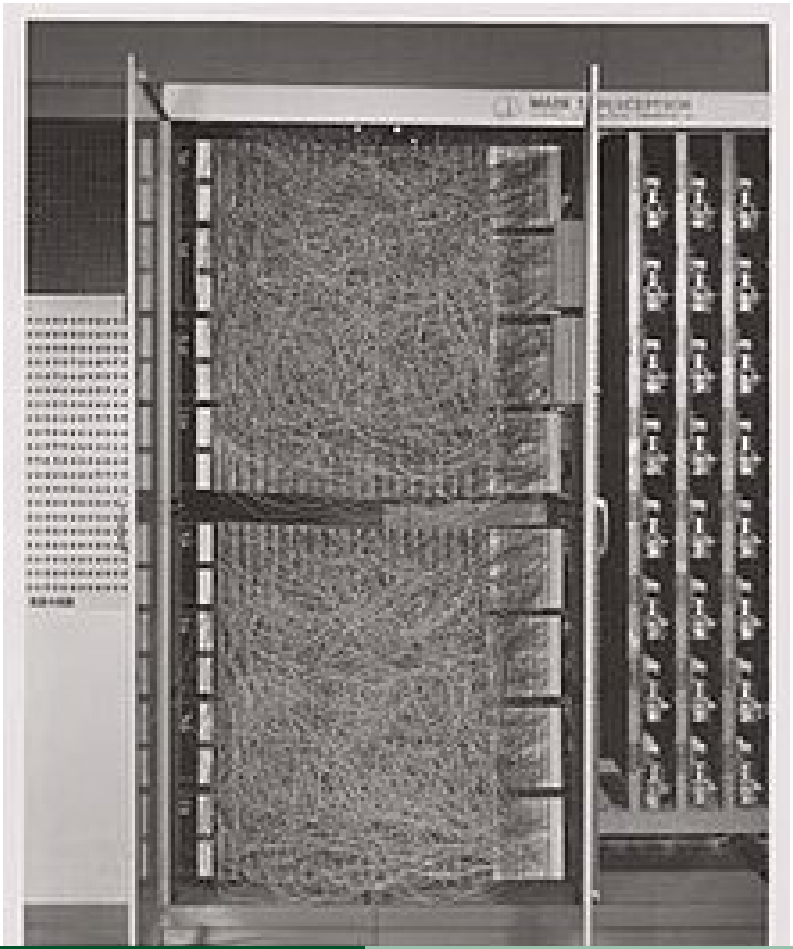
How do we teach the machine learning algorithm to come up with this decision boundary automatically?

We have a binary classification task where we have two classes, the blue diamonds and the orange triangles. So now the big question is, how does the machine learning algorithm learn this decision boundary?

We use a machine learning algorithm!

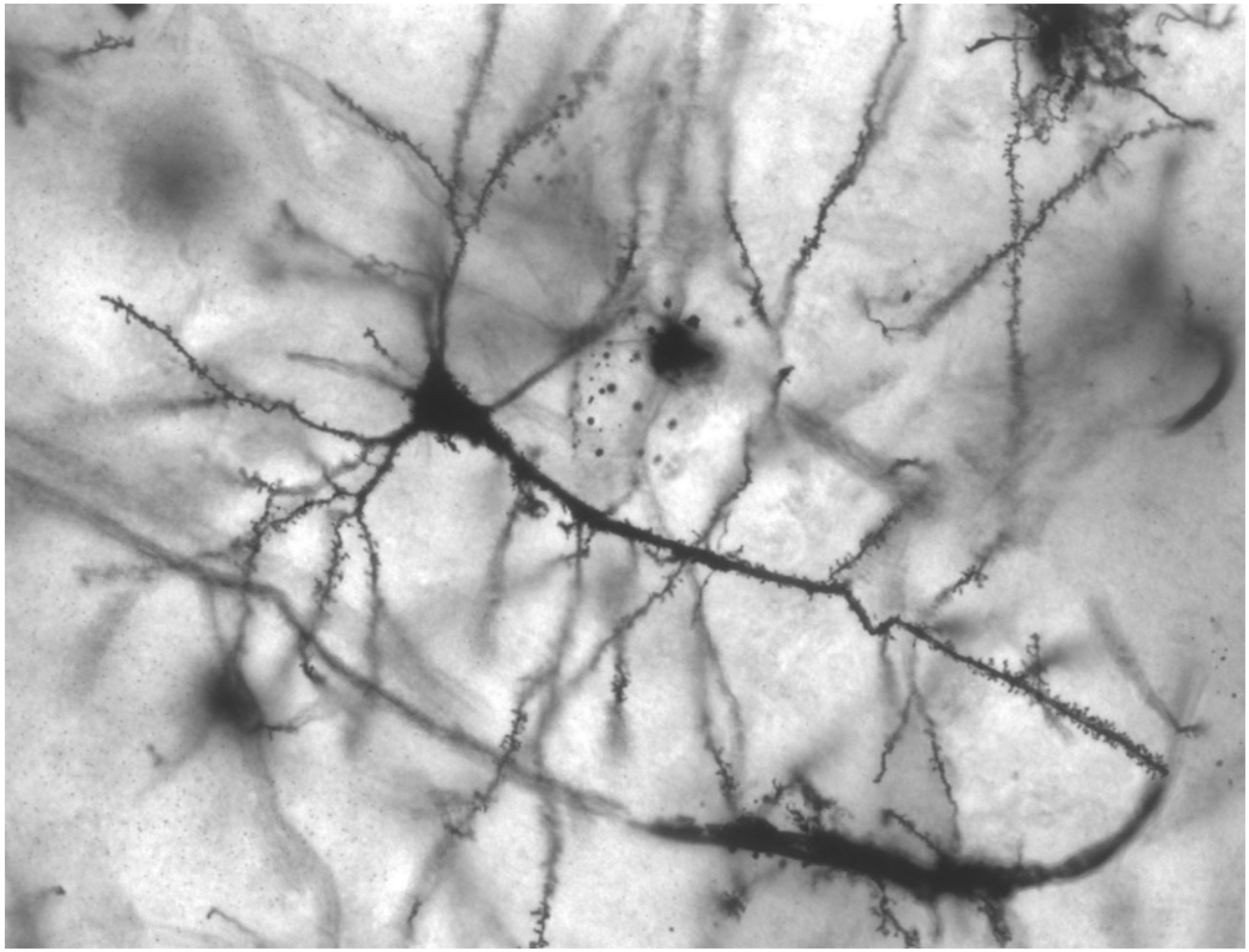
The Perceptron

So to see how this works, we will take a look at one of **the first machine learning algorithms called the perceptron**.



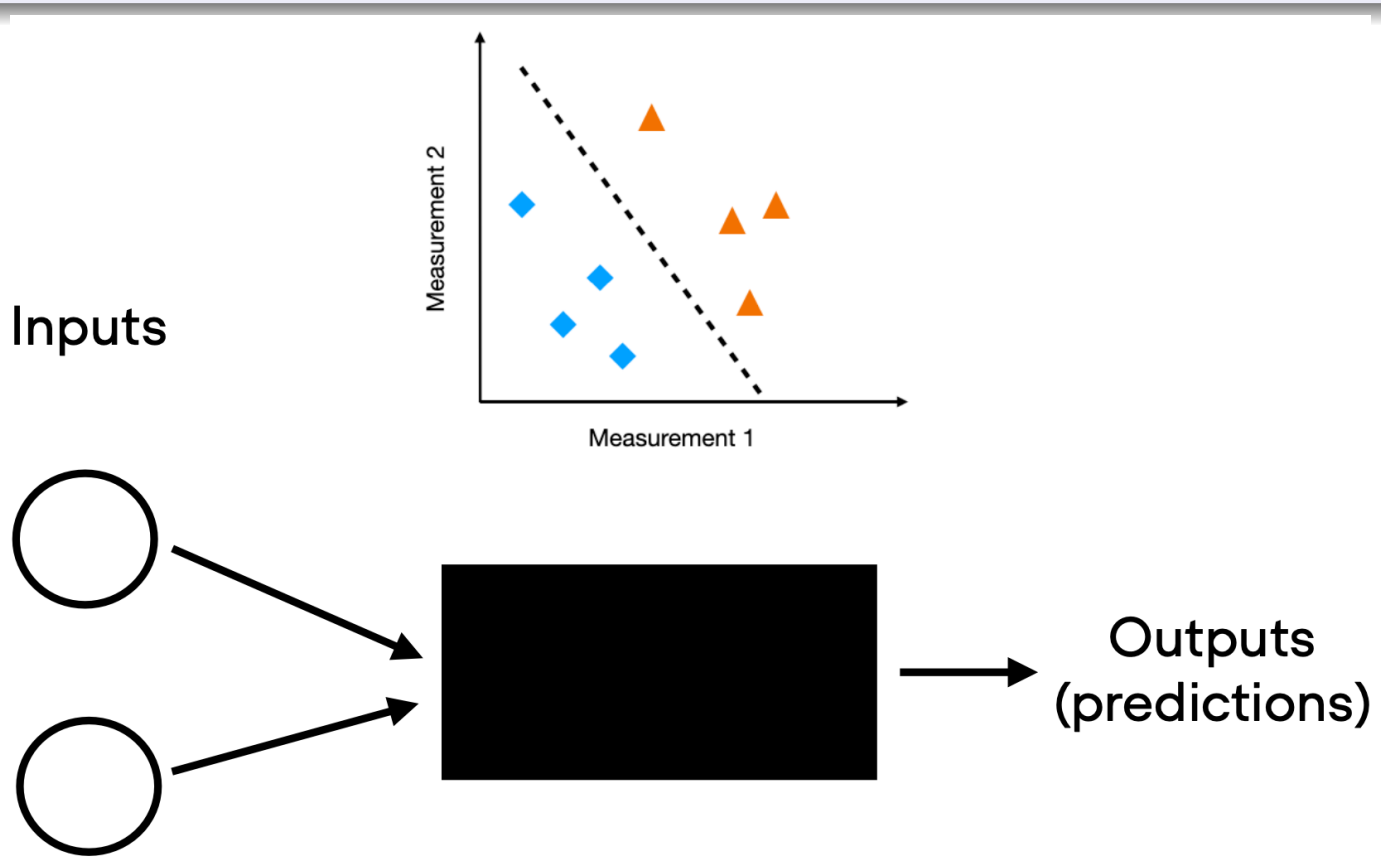
It was first implemented in hardware, it was actually a box with different wires.

The perceptron Inspired by Neurons in the Human Brain. *The motivation for the perceptron was how the human brain works.*



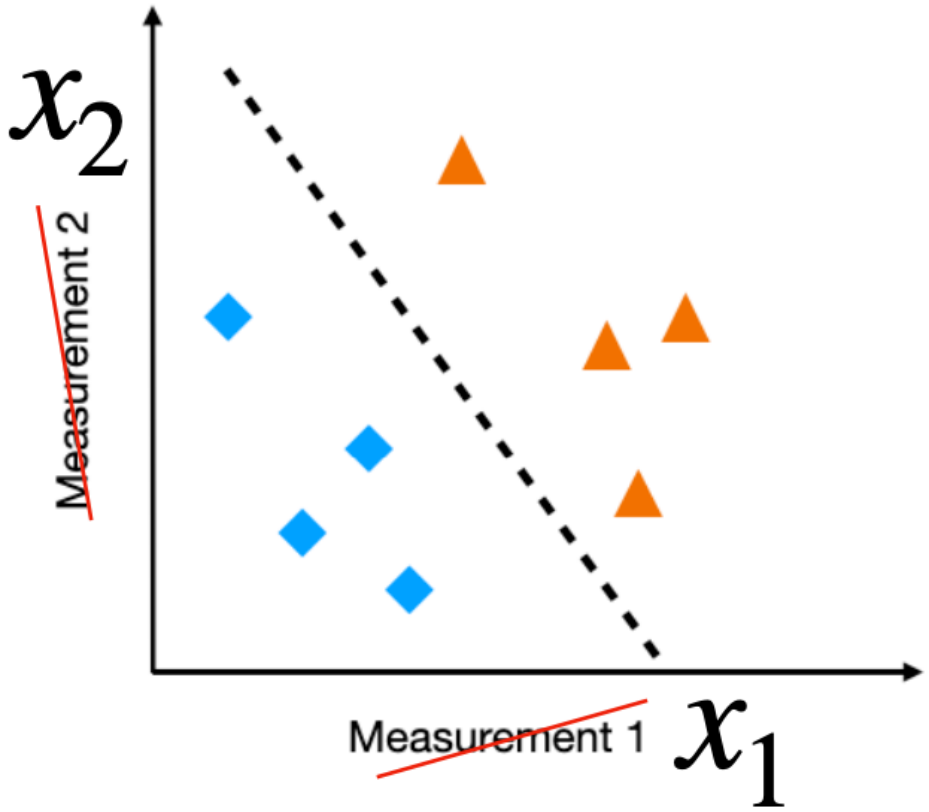
The perceptron was implemented analogous to how neurons in the human brain work?. This is not exactly how the human brain works.

The overall structure of a perceptron looks like that. We have inputs here, and **the inputs go into a black box**, which we will define later.

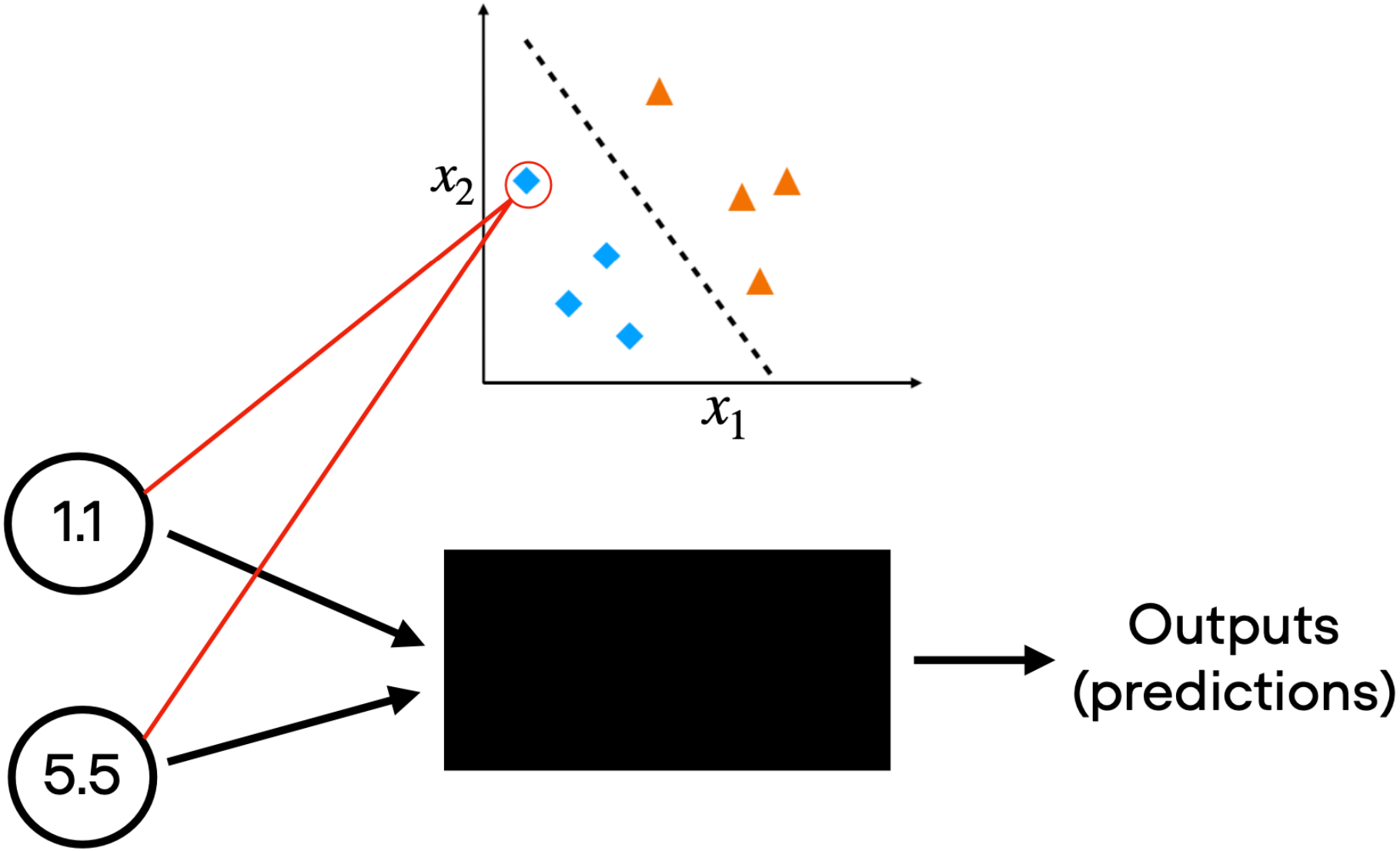


Hình 1: We have inputs, our features, our measurements, and the outputs represent our class labels. And the number of inputs depends on the number of dimensions in our data set.

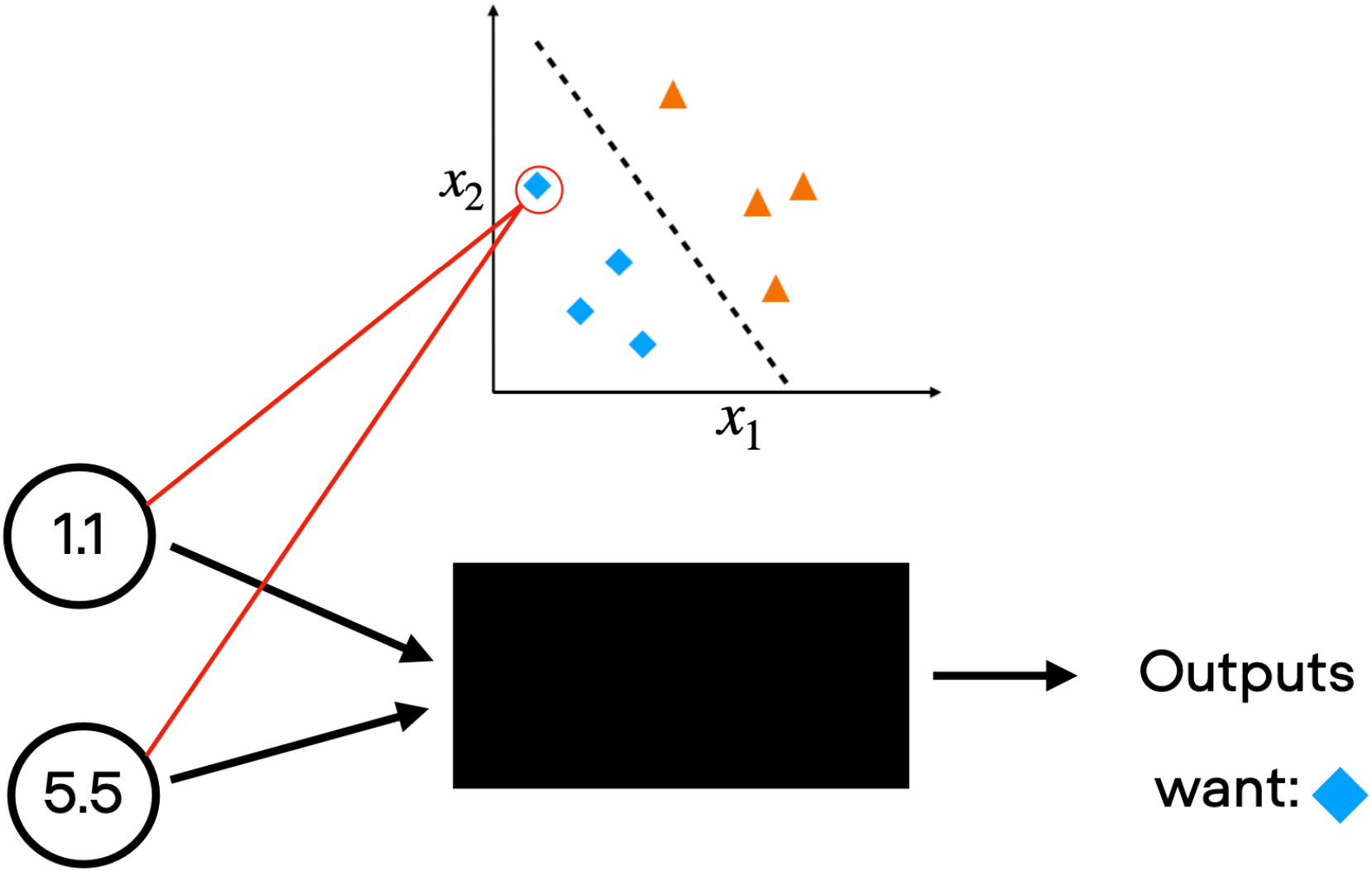
In machine learning terms, we often refer to these measurement or feature variables as x. So in our two-dimensional data set, feature one would be x_1, and feature number two would be x_2.



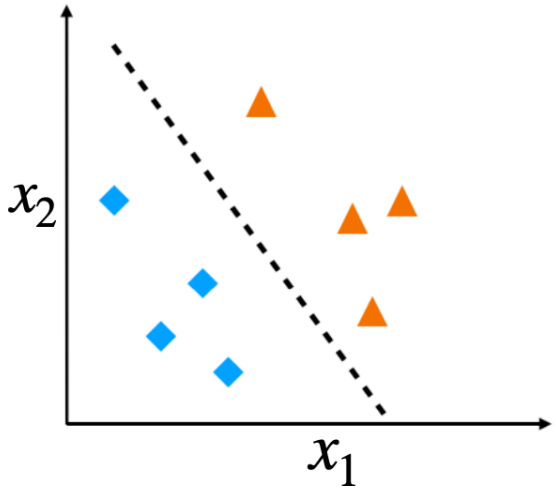
Let's pick out a particular point $(x_1 = 1.1,\ x_2 = 5.5)$ from this data set and **see how it's processed by the perceptron**.



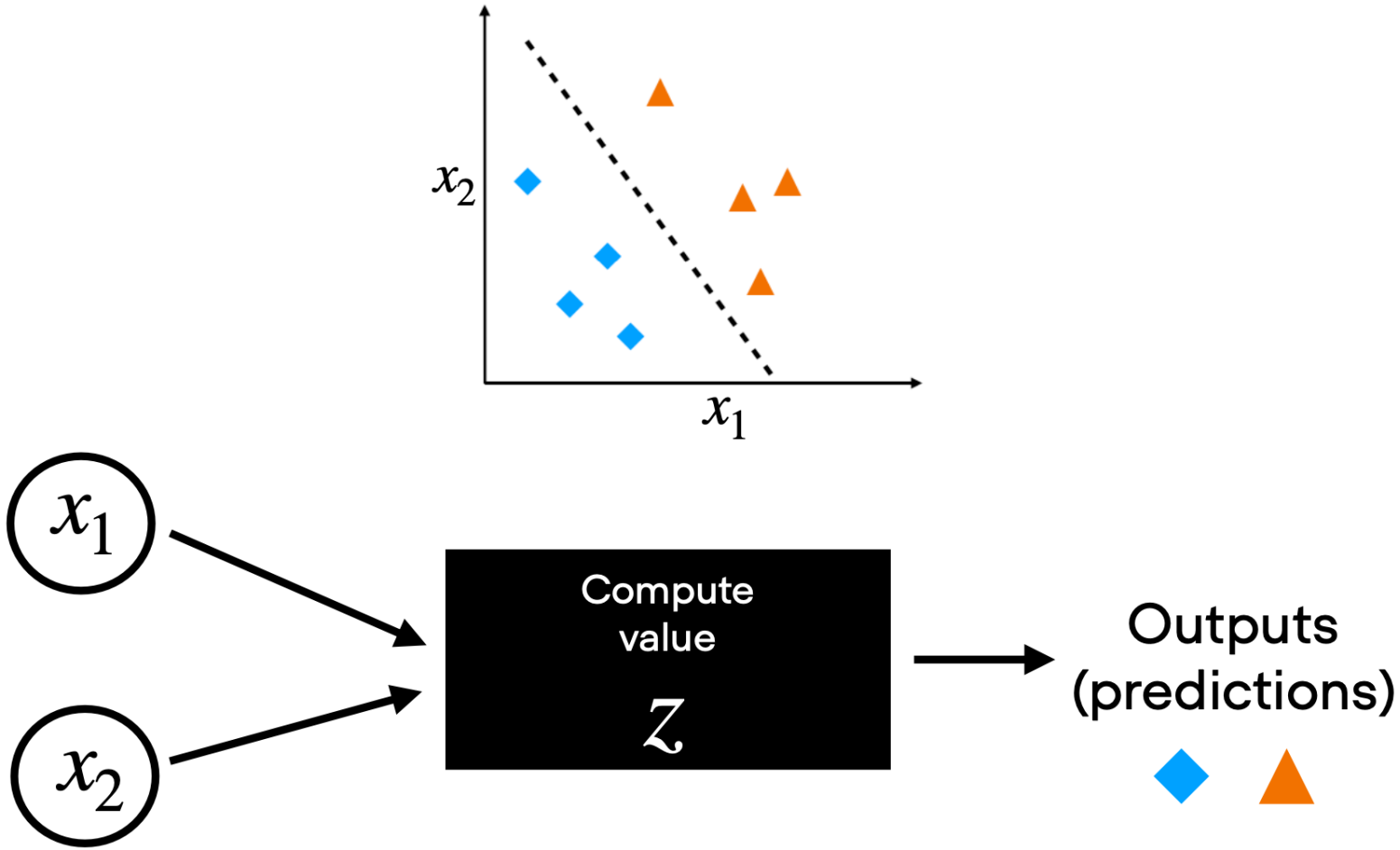
In this particular case, since that's **our training data set, we already know the true answer.**



In real life, when we have a prediction problem, **we apply it to new data** where we don't know the answer yet, and it should come up with the predictions.

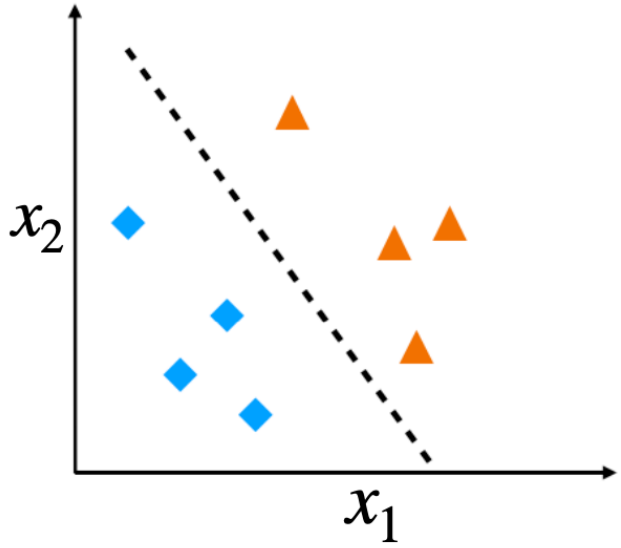


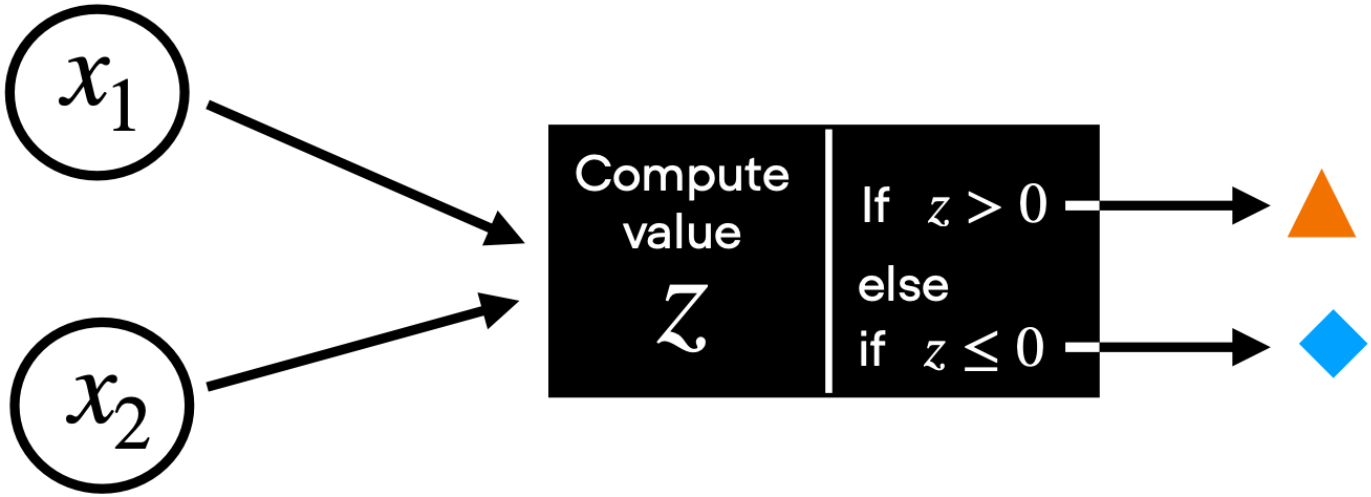
Now let's take a look at this **black box** (xem hình 1) and see what happens inside. So inside the perceptron computes a so-called Z value, which we also sometimes call the **net input**.



Our decision rule

Consider we are given this value Z, and then we apply a **threshold** to it.

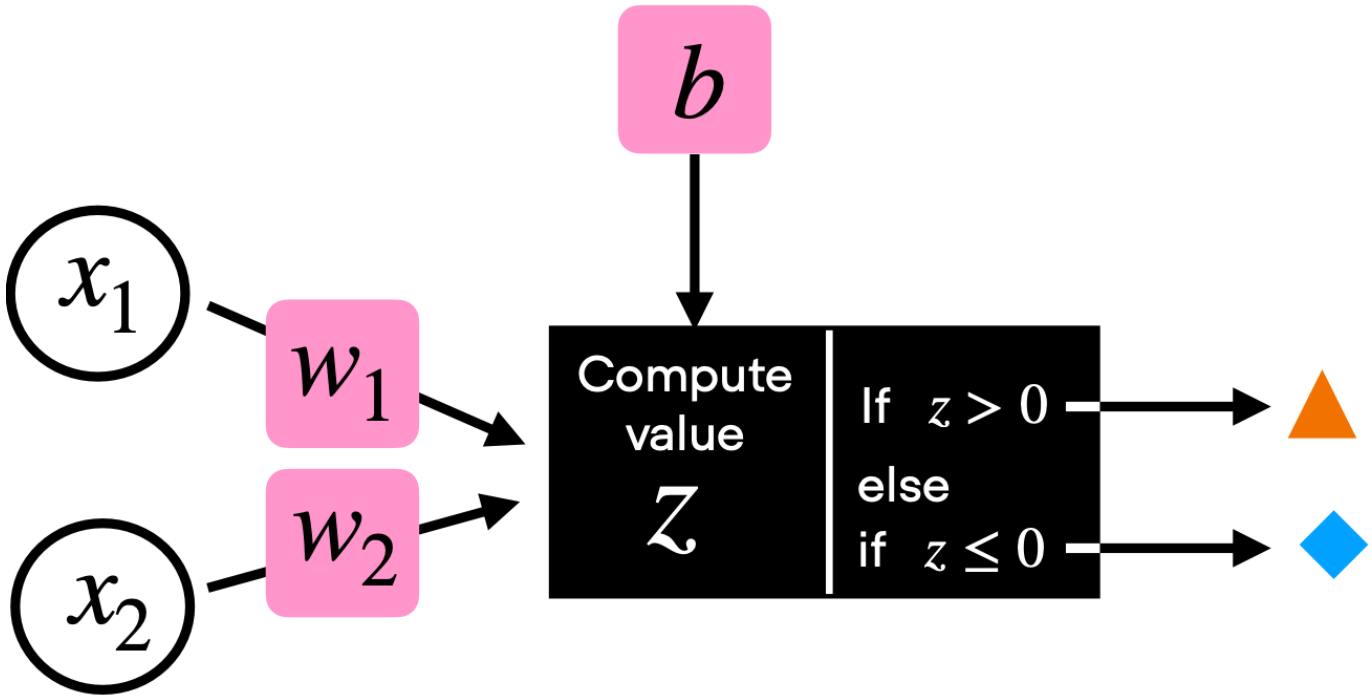




Model parameters to compute value Z

How do we compute this value Z though?

We will involve a *few parameters*. We call them the **model parameters**. And these are essentially **the things that the perceptron learns**.

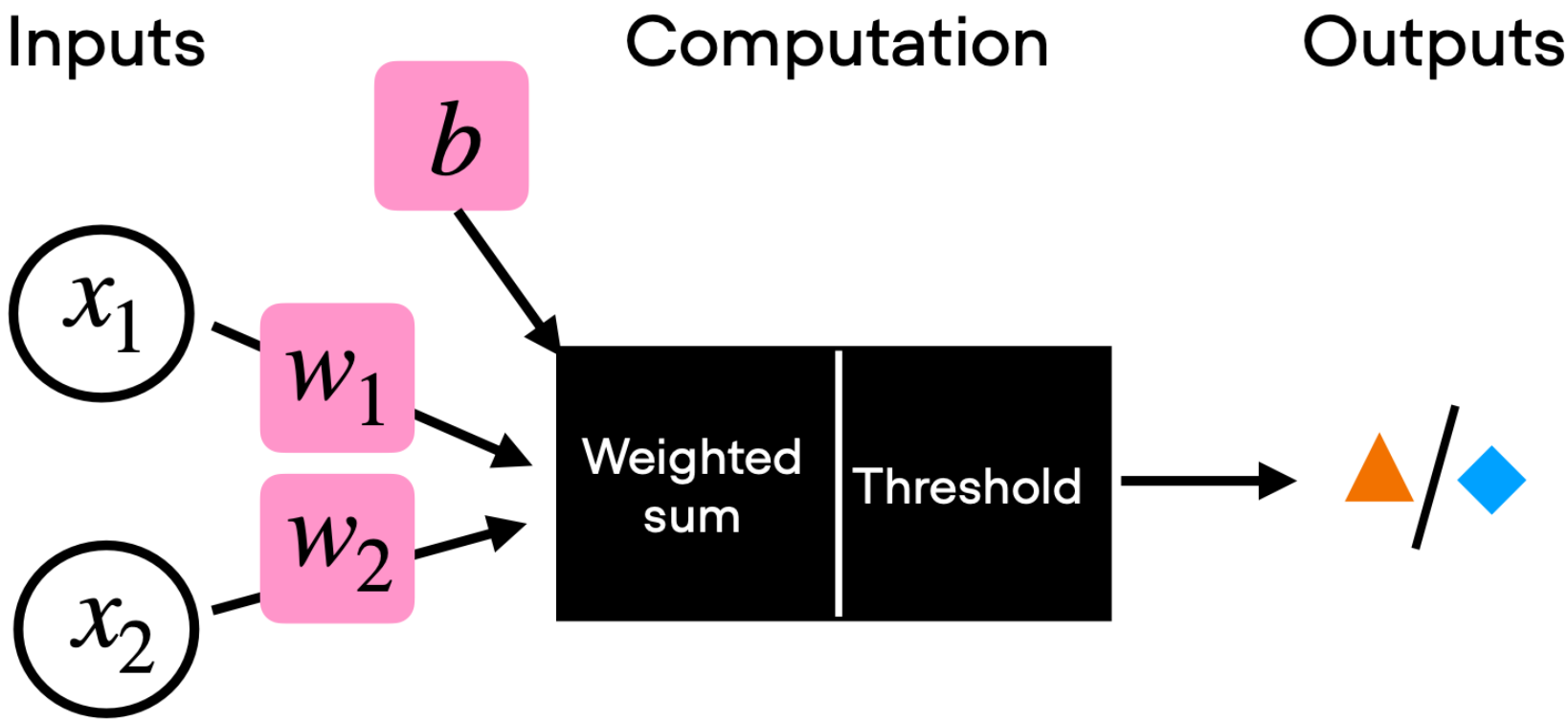


Model weights w_1, w_2 and **bias unit** b are values that are learned from the training data set. Each input feature value has a coressponding model weight.

In many real world cases, we have higher dimensional data sets. So we can extend the above formula and go up to m features

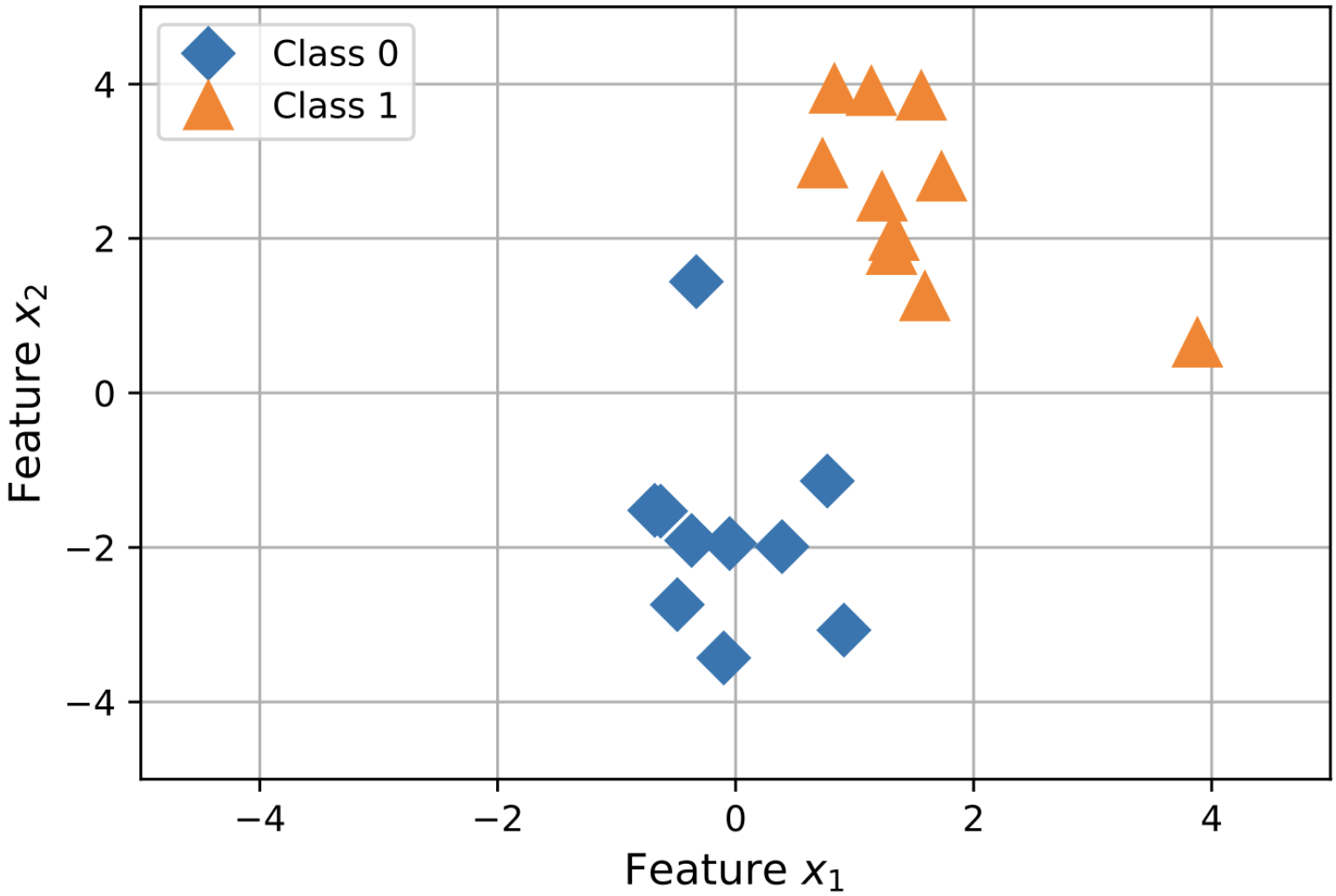
$$Z = x_1 \times w_1 + x_2 \times w_2 + \cdots + x_m \times w_m + b = b + \sum_{i=1}^{m} x_i \times w_i$$

Perceptron Summary for Making Predictions



how the perceptron works.

First of all, we need a training data set to train our perceptron and to learn the model parameters w, b. Start with a Training Dataset:



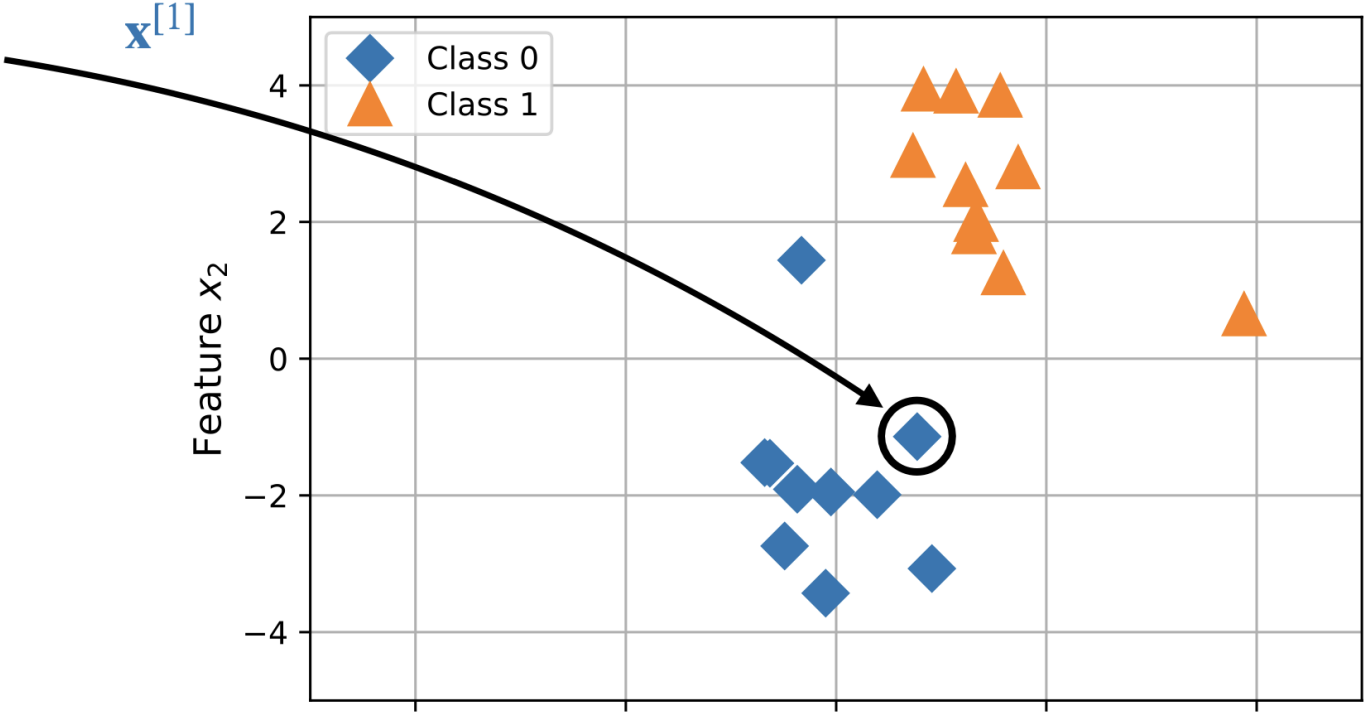
Định nghĩa tập dữ liệu huấn luyện:

$$\mathcal{D} = \left(\left\langle \mathbf{x}^{[1]}, y^{[1]} \right\rangle, \left\langle \mathbf{x}^{[2]}, y^{[2]} \right\rangle, \ldots, \left\langle \mathbf{x}^{[n]}, y^{[n]} \right\rangle\right)$$

$$\mathbf{x}^{[1]} = \left[x_1^{[1]}, x_2^{[1]}\right]$$

x_1	x_2	y
0.77	1.14	0
-0.33	1.44	0
0.91	-3.07	0
-0.37	-1.91	0
-0.63	-1.53	0
0.39	-1.99	0
...	...	...

Training example $\mathbf{x}^{[1]}$

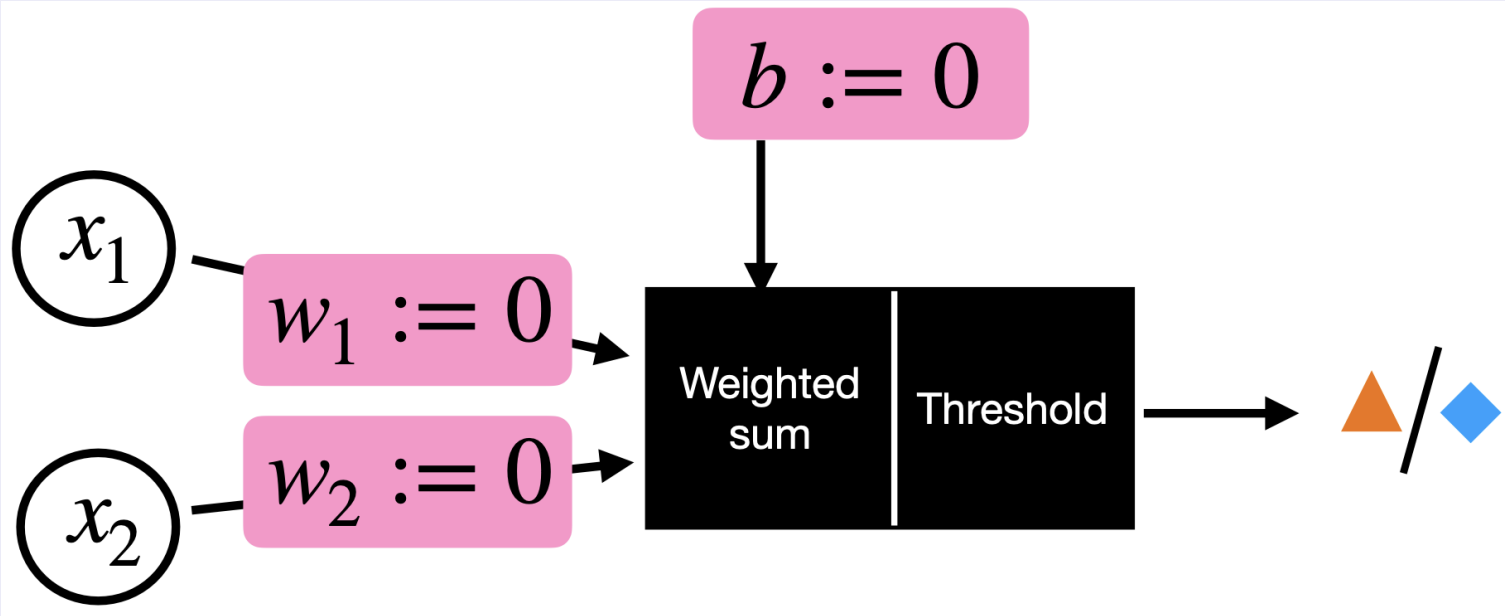


x_1	x_2	y
0.77	1.14	0
-0.33	1.44	0
0.91	-3.07	0
-0.37	-1.91	0
-0.63	-1.53	0
0.39	-1.99	0
...	...	...
1.33	2.03	1

$$\mathbf{x}^{[20]} = \left[x_1^{[20]},\ x_2^{[20]}\right].$$

How does the perceptron learn the model parameters?

1. Define training set
2. Initialize model weights and bias to zero

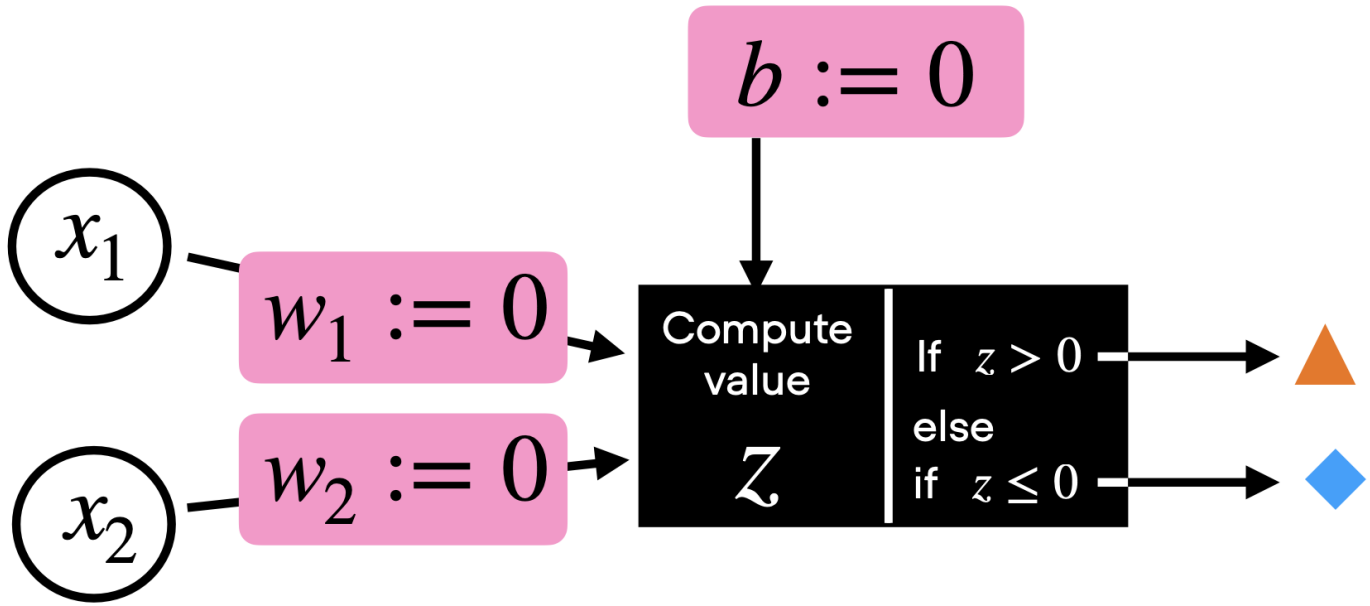


3. For every training epoch:
   - For every training example $\langle \mathbf{x}^{[i]}, y^{[i]} \rangle \in \mathcal{D}$:
     1. Make a prediction
     2. Compute the error
     3. Update the weights based on the error

Make a prediction

Tính Weighted sum Z cho từng training example.

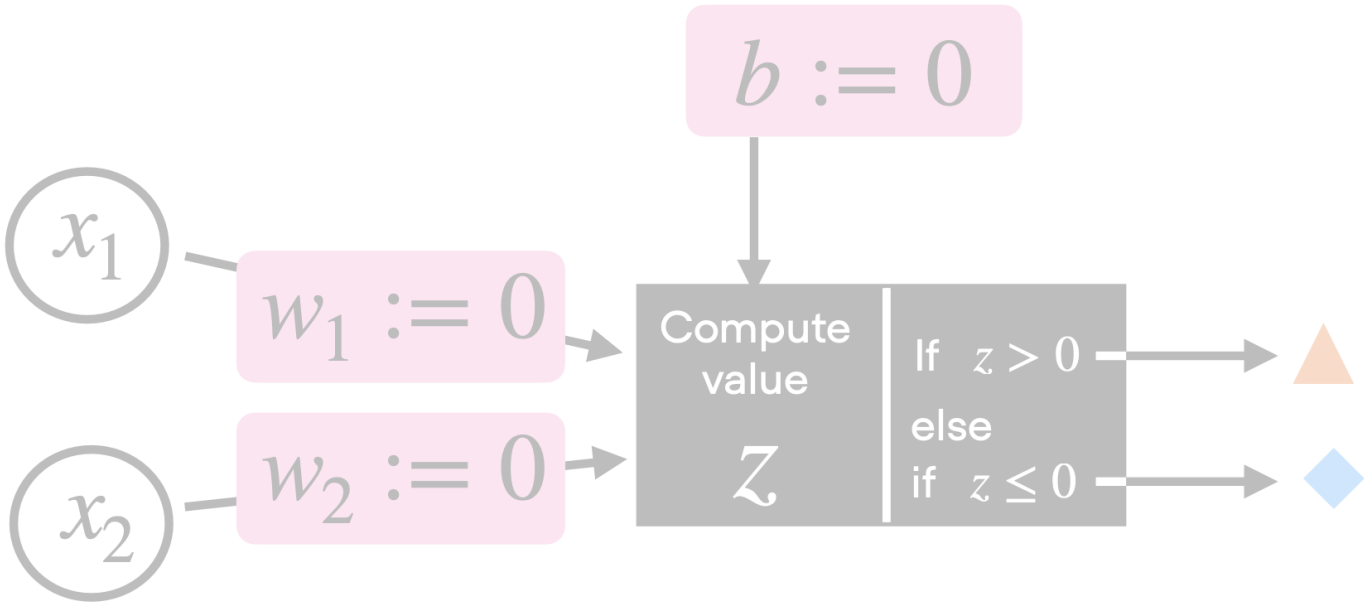
$$x_1 \times w_1 + x_2 \times w_2 + b = z$$



Make a prediction

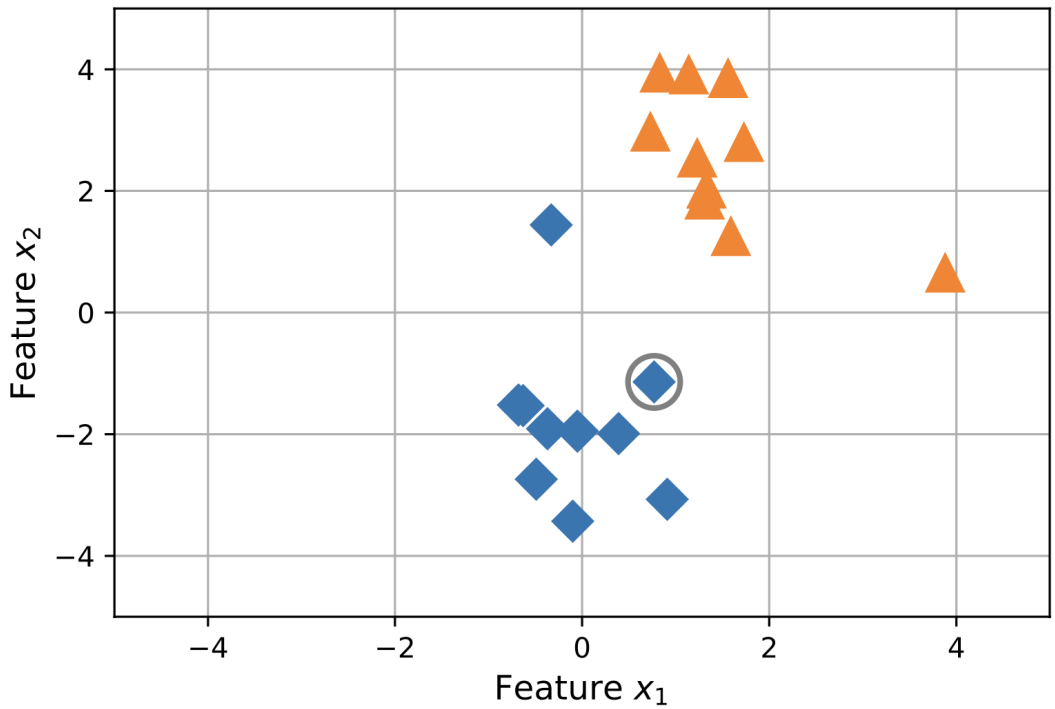
Do ban đầu khởi tạo trọng số model ở bước 2 bằng 0 nên $Z^{[1,10]} = 0$.

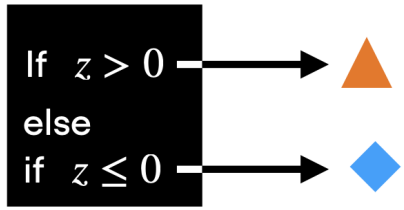
$$x_1 \times 0 + x_2 \times 0 + 0 = 0$$



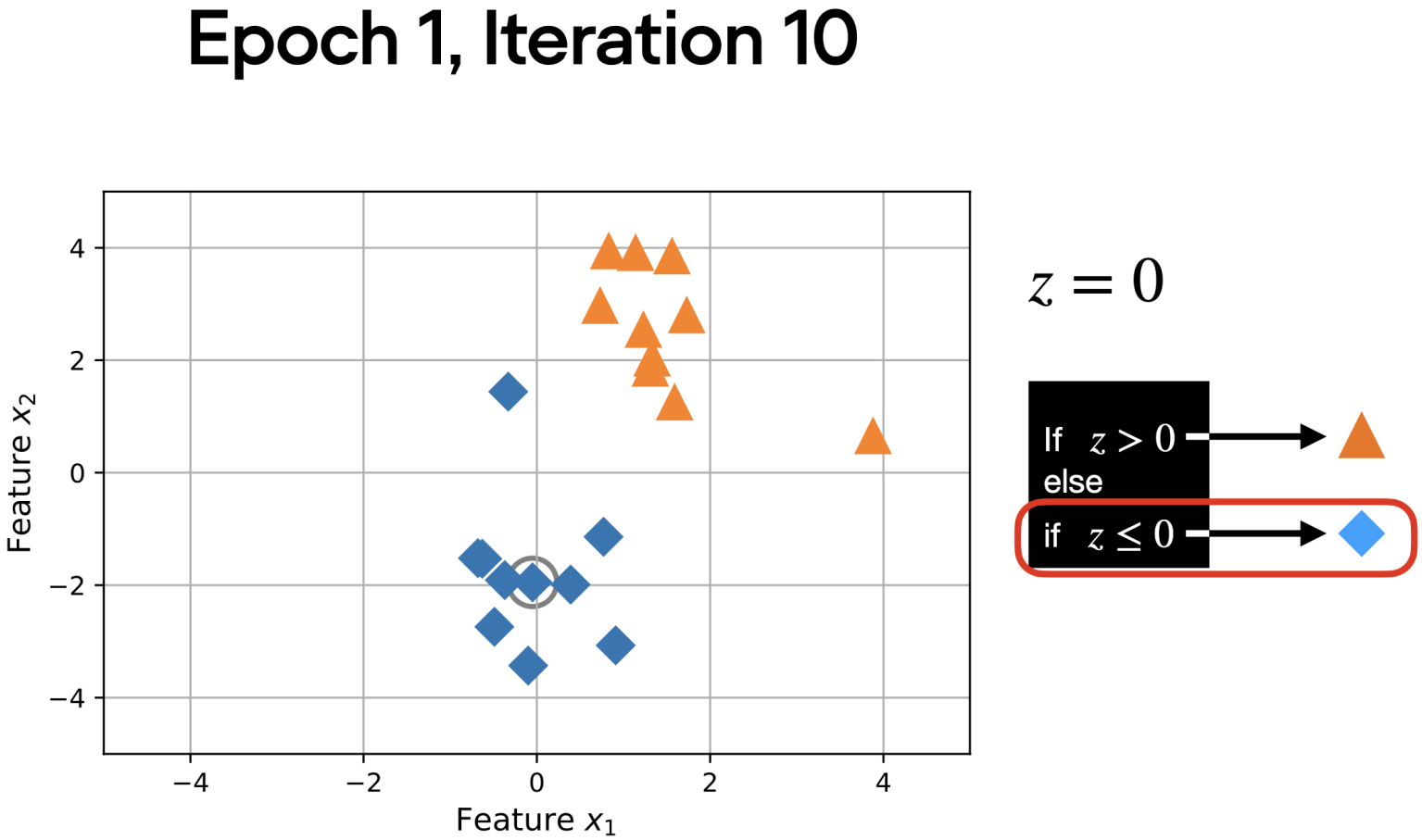
Vì $Z^{[1,10]} \leq 0$ tương ứng từ Training example 1 đến Training example 10 sẽ dự đoán là Blue diamond $\Diamond$ nên sẽ không có cập nhật trọng số dựa trên error. Bước 3.2 Compute the error, 3.3 Update the weights based on the error không có thay đổi.

Epoch 1, Iteration 1



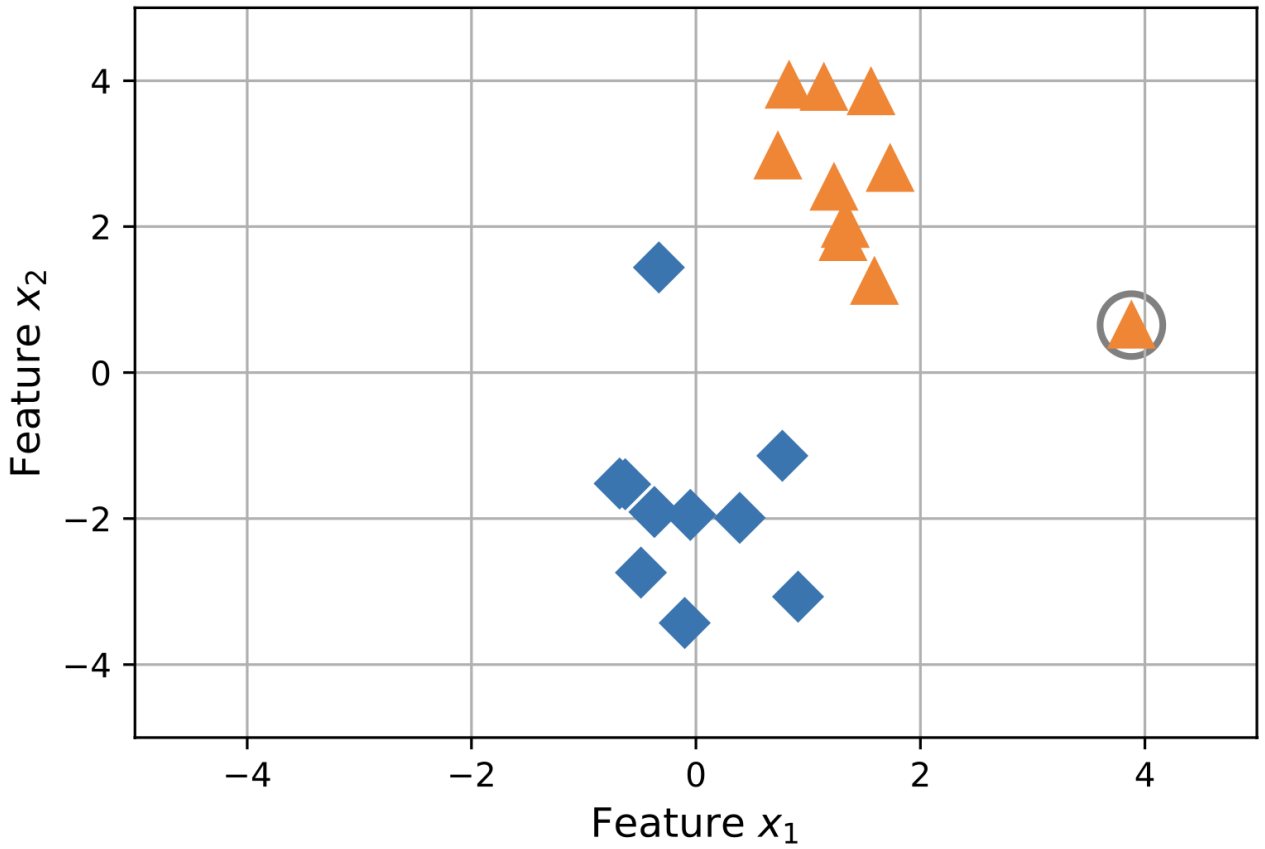


Make a prediction



Thông thường chúng ta nên shuffle dataset để cập nhật trọng số sớm hơn.

Epoch 1, Iteration 11



$z = 0$

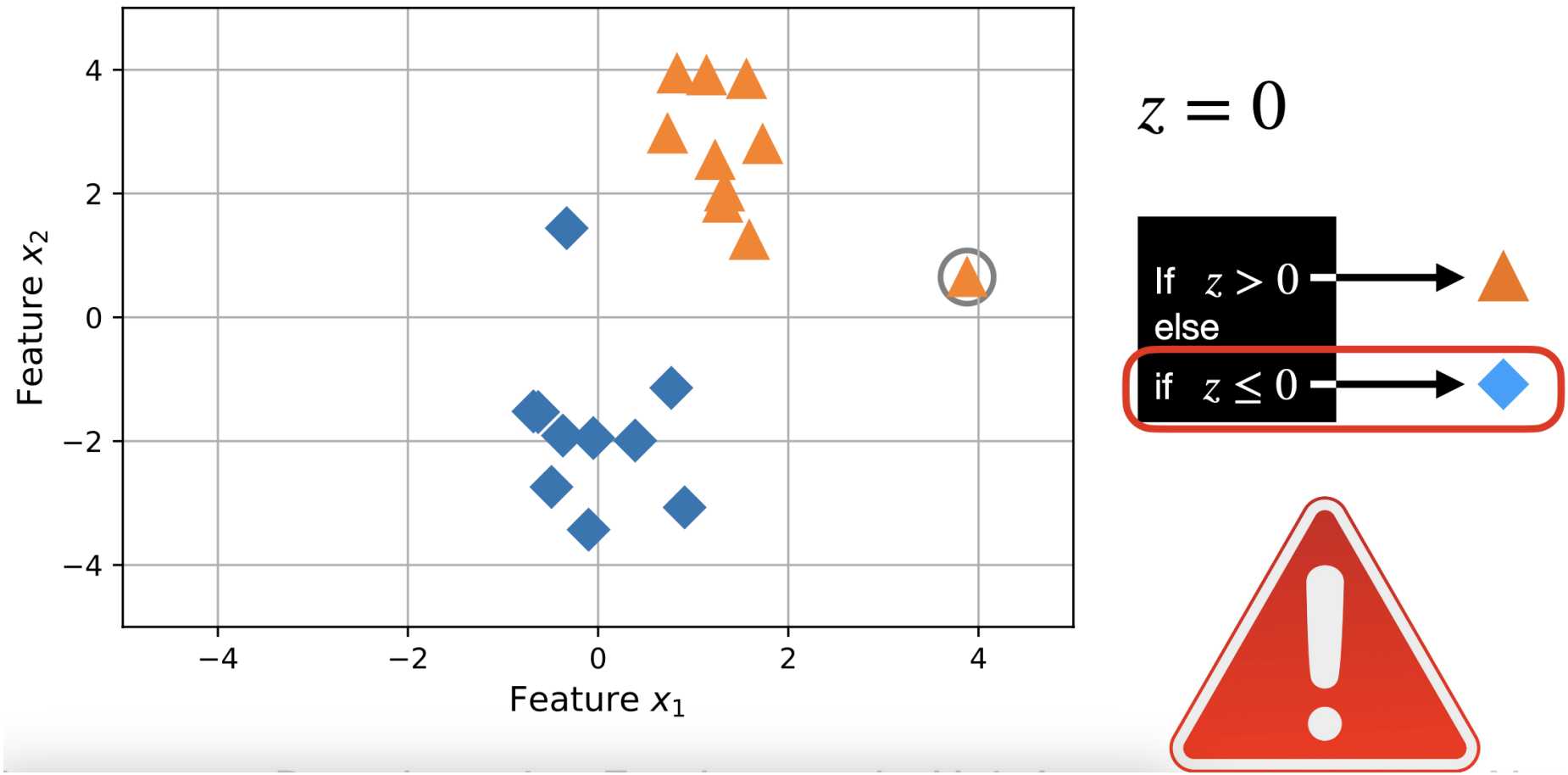
If $z > 0$

else

if $z \leq 0$

Bước 3.1. Making prediction

Epoch 1, Iteration 11

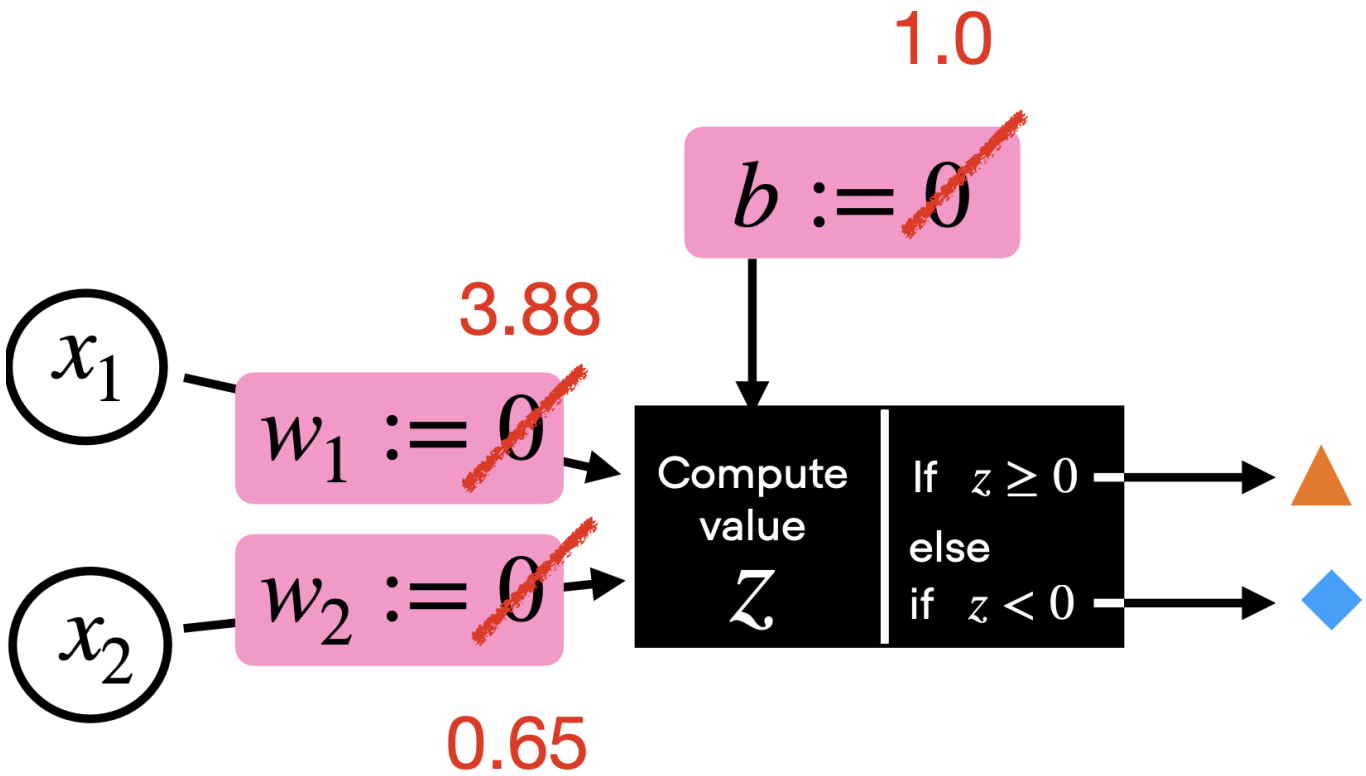


Bước 3.1. Tại Epoch 1, training example thứ 11 sẽ dự đoán sai.

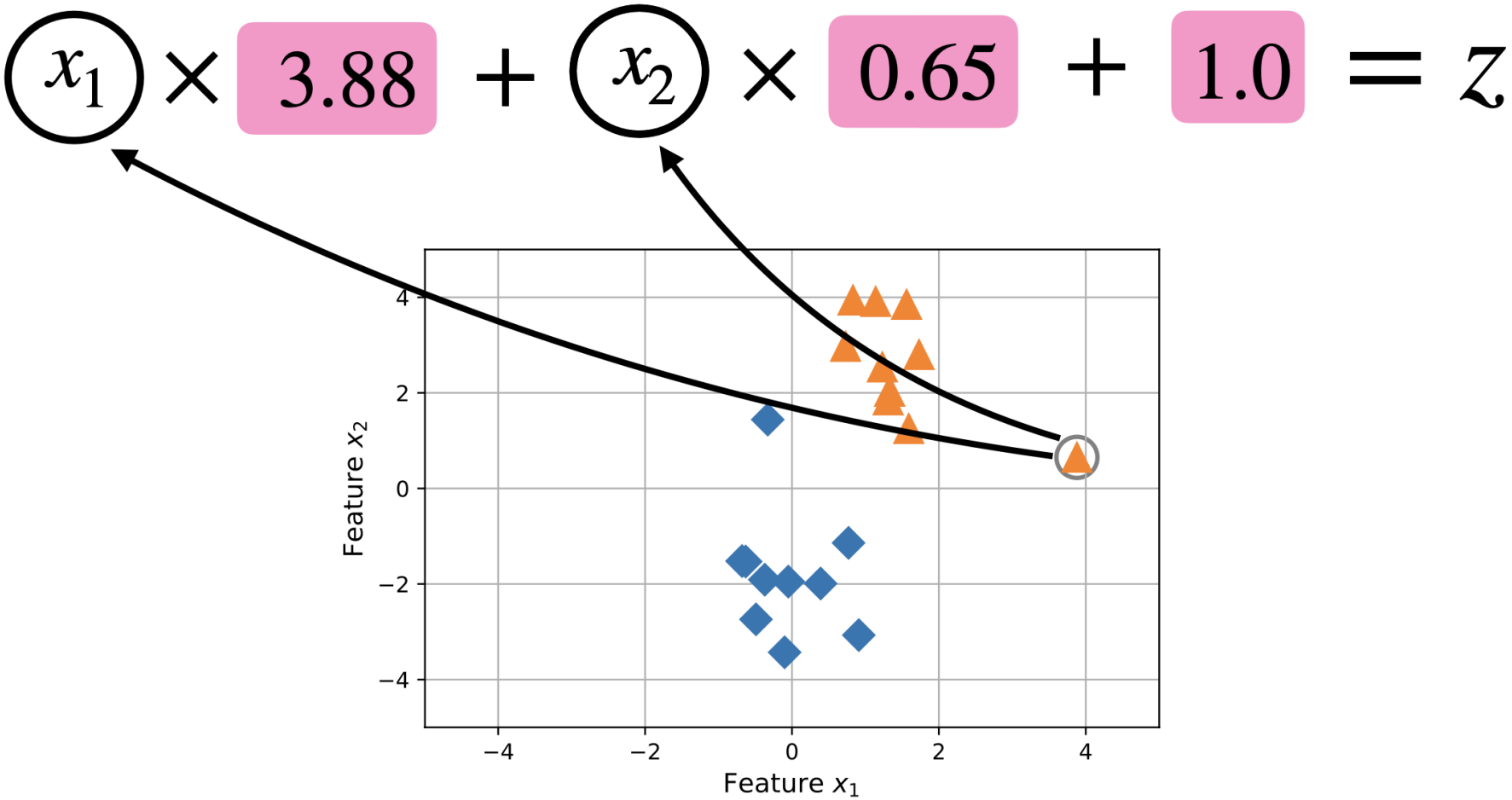
Epoch 1, Iteration 11

Bước 3.2, 3.3

Chúng ta tiến hành tính lại error và cập nhật lại model weights.
Công thức tính error và cập nhật weights sẽ được đề cập sau.

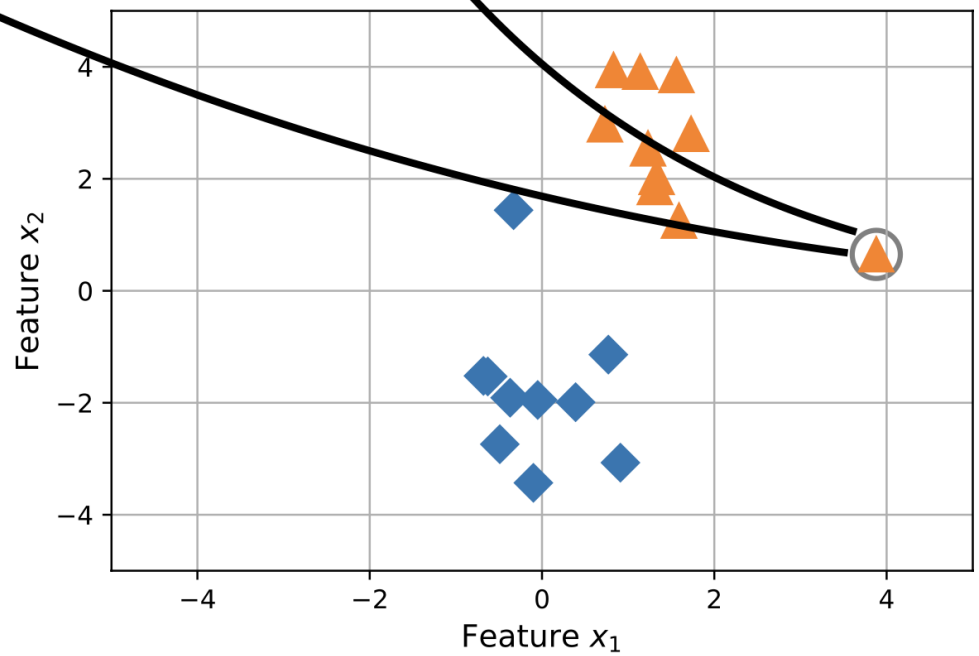


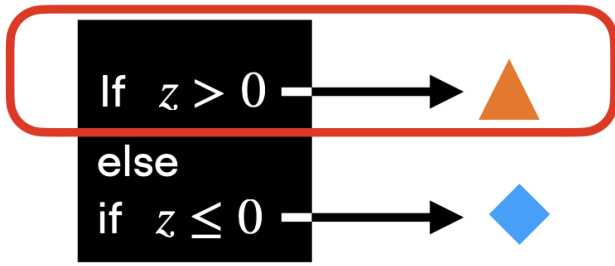
Epoch 1, Iteration 11



Epoch 1, Iteration 11

$$0.83 \times 3.88 + 3.94 \times 0.65 + 1.0 = 6.78$$

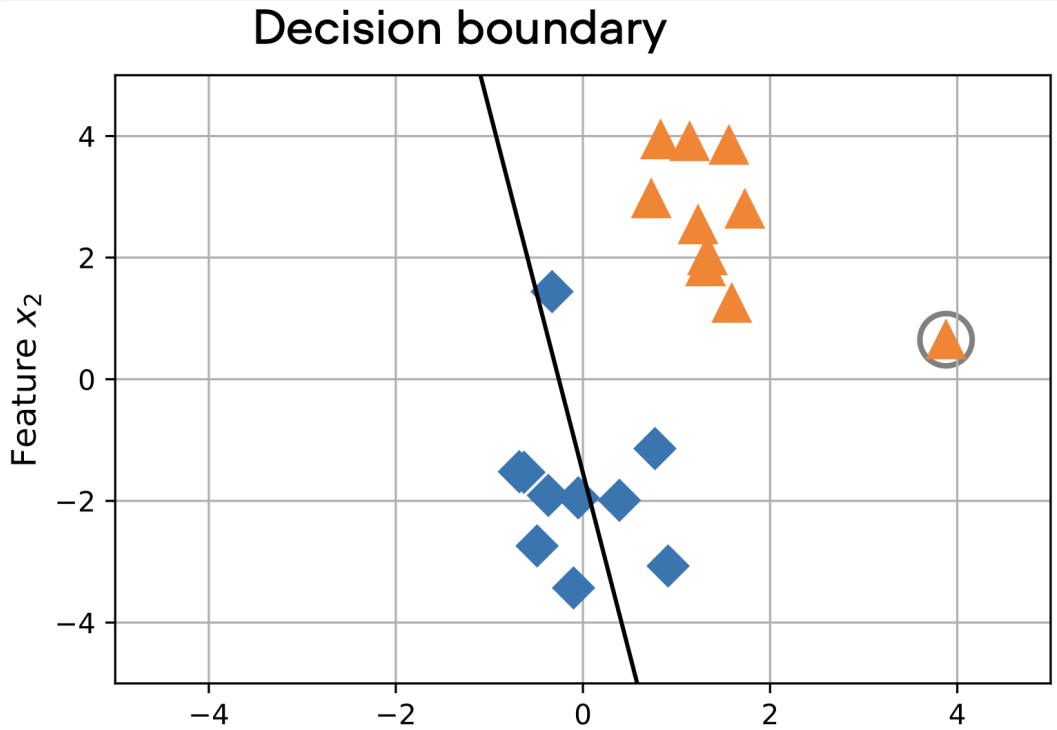




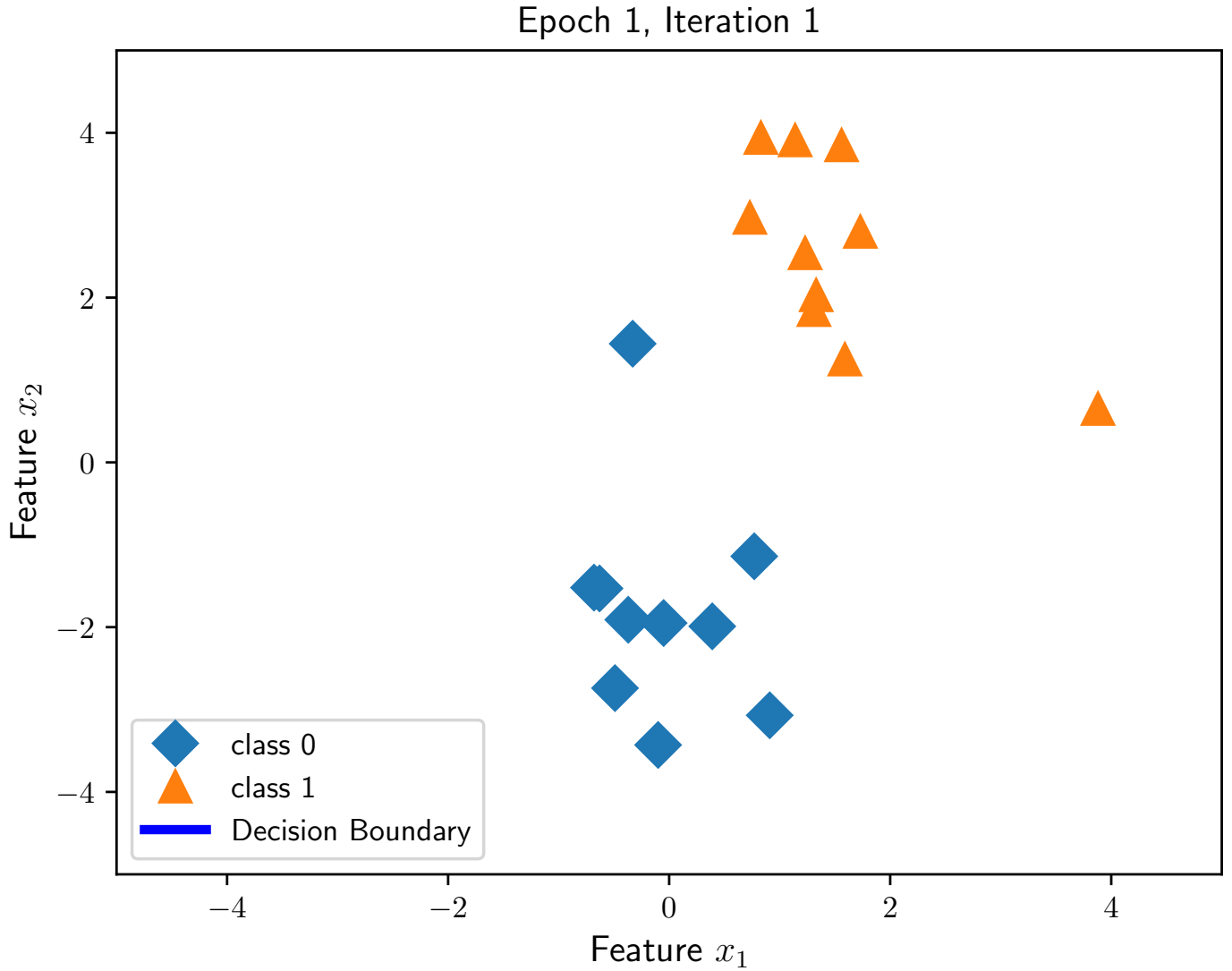
Decision boundary

Sau khi cập nhật trọng số ở Epoch 1, Iteration 11, chúng ta thu được **Decision boundary**

$$x_2 = \frac{-b - x_1 \times w_1}{w_2}$$



Lặp lại Bước (3) cho đến khi 100% các ví dụ được phân loại chính xác.



Code Python thuật toán Perceptron I

```
1  import pandas as pd
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  df=pd.read_csv("perceptron_toydata-truncated.txt", sep="\t")
6  X_train=df[["x1","x2"]].values
7  y_train=df['label'].values
8
9  class Perceptron(object):
10     def __init__(self, num_features) -> None:
11         self.num_features = num_features
12         self.weights = [0 for _ in range(num_features)]
13         self.bias = 0.0
14     def forward(self, x):
15         weighted_sum_z = self.bias
16         for i,_ in enumerate(self.weights):
17             weighted_sum_z += self.weights[i] * x[i]
18         if weighted_sum_z > 0:
19             return 1
20         else:
```

Code Python thuật toán Perceptron II

```
21              return 0
22      def update(self, x, true_y):
23          prediction = self.forward(x)
24          error = true_y - prediction
25          self.bias += error
26          for i, _ in enumerate(self.weights):
27              self.weights[i] += x[i] * error
28          return error
29
30  def train(model, all_x, all_y, epochs):
31      for epoch in range(epochs):
32          error_count=0
33          for x, y in zip(all_x, all_y):
34              error = model.update(x, y)
35              error_count += abs(error)
36          print(f'''Epoch {epoch+1}, error {error_count}
37      Model parameter:
38          weights: {model.weights}
39          bias: {model.bias}''')
40
```

Code Python thuật toán Perceptron III

```
41  def compute_accuracy(model, all_x, all_y):
42      correct = 0.0
43      for x, y in zip(all_x, all_y):
44          correct += int(model.forward(x) == y)
45      return correct/len(all_y)
46
47  ppn= Perceptron(num_features=2)
48  num_epoch=6
49  train(model=ppn, all_x=X_train, all_y=y_train, epochs=num_epoch)
50  train_acc = compute_accuracy(model=ppn, all_x=X_train, all_y=y_train)
51  print(f"Accuracy of Perceptron model after {num_epoch} epochs:
        {train_acc*100}%")
52
53  def plot_boundary(model):
54      w_1, w_2 = model.weights
55      bias = model.bias
56      x_min, x_max = -10, 10
57      y_min = (-w_1*x_min - bias)/w_2
58      y_max = (-w_1*x_max - bias)/w_2
59      return [x_min, x_max], [y_min, y_max]
```

Code Python thuật toán Perceptron IV

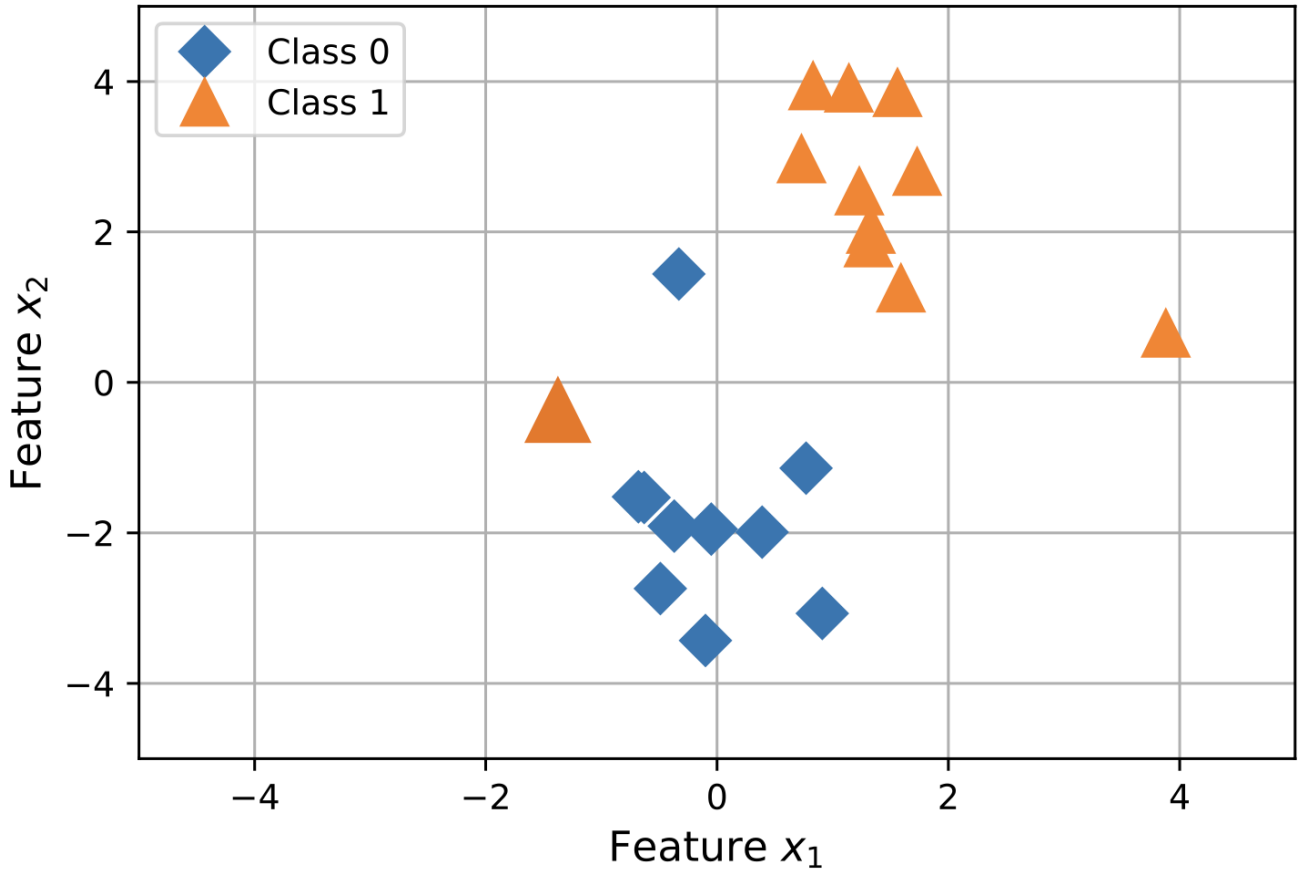
```
60
61  plt.plot(X_train[y_train==0,0],
62           X_train[y_train==0,1],
63           marker="D",
64           markersize=10,
65           linestyle="",
66           label="class 0"
67  )
68  plt.plot(X_train[y_train==1,0],
69           X_train[y_train==1,1],
70           marker="^",
71           markersize=10,
72           linestyle="",
73           label="class 1"
74  )
75  x_axis, y_axis = plot_boundary(ppn)
76  plt.plot(x_axis, y_axis, color="blue", linewidth=3, label="Decision Boundary")
77  plt.xlabel("Feature $x_1$", fontsize=12)
78  plt.ylabel("Feature $x_2$", fontsize=12)
79  plt.xlim([-5, 5])
```

Code Python thuật toán Perceptron V

```
80  plt.ylim([-5, 5])
81  plt.legend(loc=3)
82  plt.show()
```

Hạn chế của của Perceptron

Các lớp phải được tách rời theo dạng tuyến tính. Trong trường hợp các điểm dữ liệu phân bố như sau sẽ không vẽ được đường thẳng Decision boundary.



Sử dụng **test set** để đánh giá mô hình

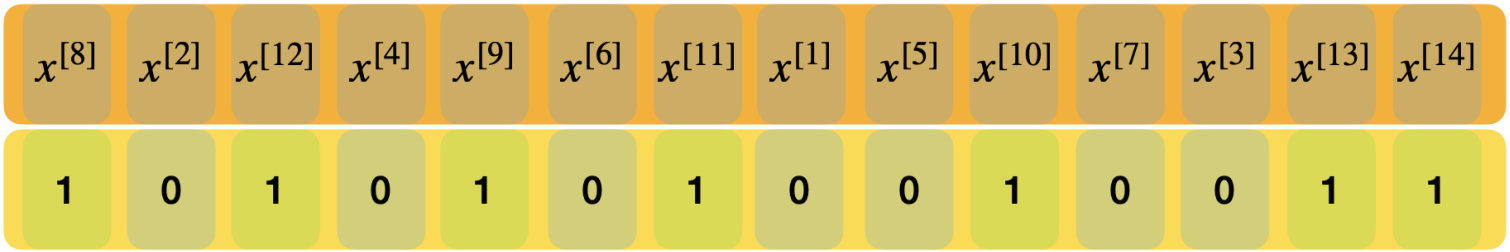
One of the big fundamentals of model evaluation is to use separate data sets for training and evaluation. So consider we are given a training dataset for classification.

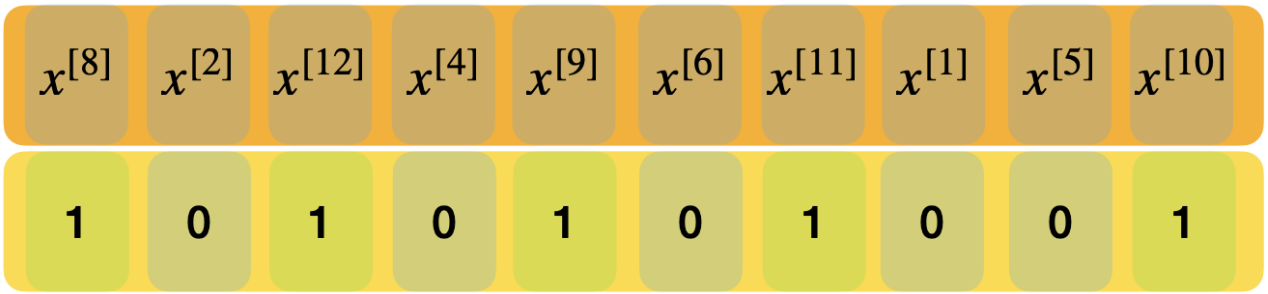
$x^{[1]}$	$x^{[2]}$	$x^{[3]}$	$x^{[4]}$	$x^{[5]}$	$x^{[6]}$	$x^{[7]}$	$x^{[8]}$	$x^{[9]}$	$x^{[10]}$	$x^{[11]}$	$x^{[12]}$	$x^{[13]}$	$x^{[14]}$
0	0	0	0	0	0	0	1	1	1	1	1	1	1

Dataset for classification: Features , Labels

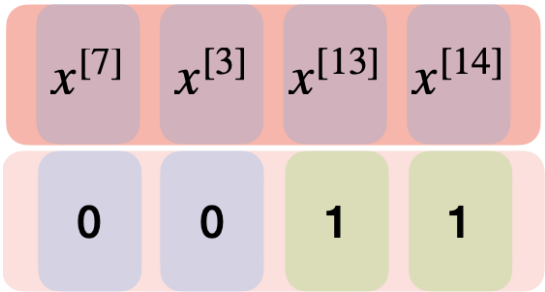
Sử dụng **test set** để đánh giá mô hình

Trước tiên, chúng ta cần phải thực hiện Data shuffling để tránh trường h





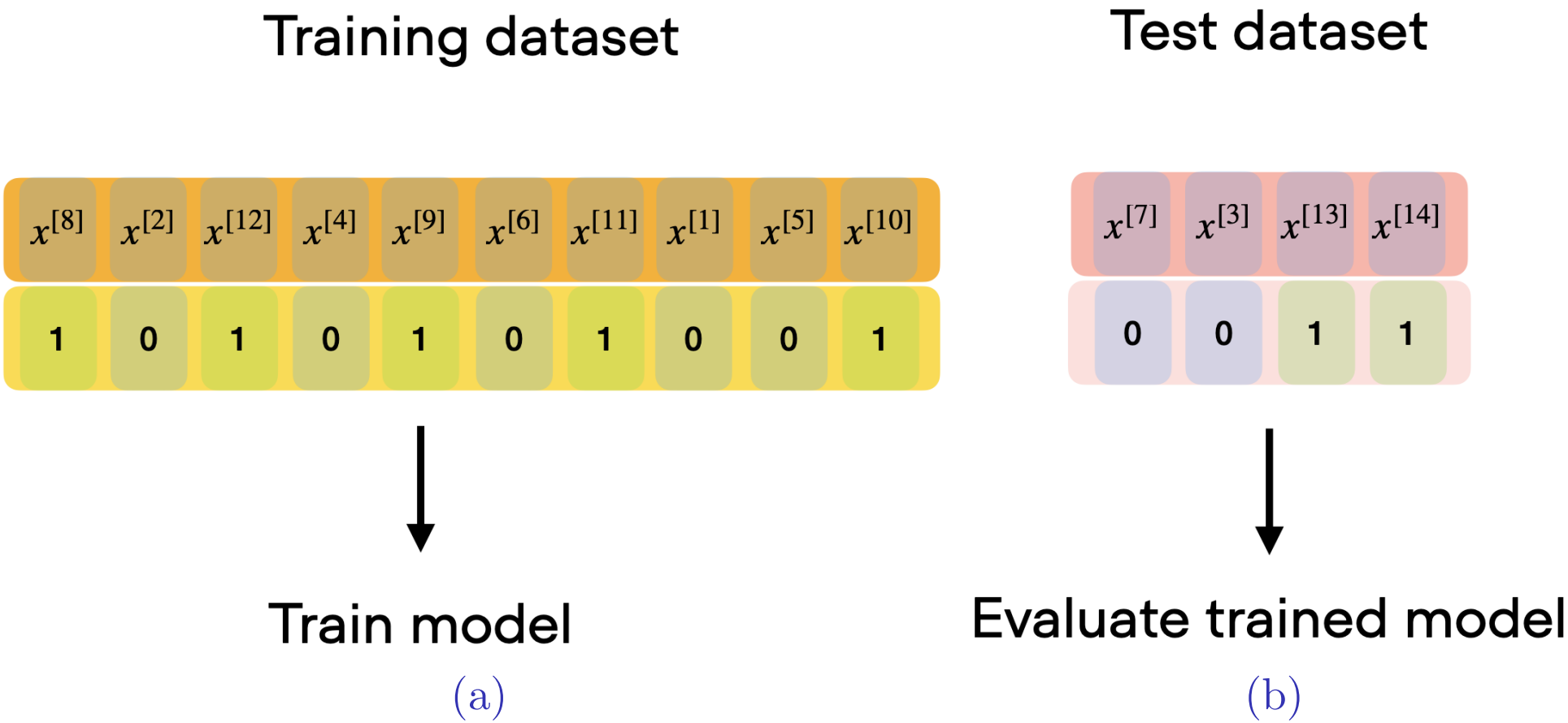
(a) Training dataset



(b) Test dataset

Hình 2: Chia làm 2 tập huấn luyện và test để đánh giá mô hình.

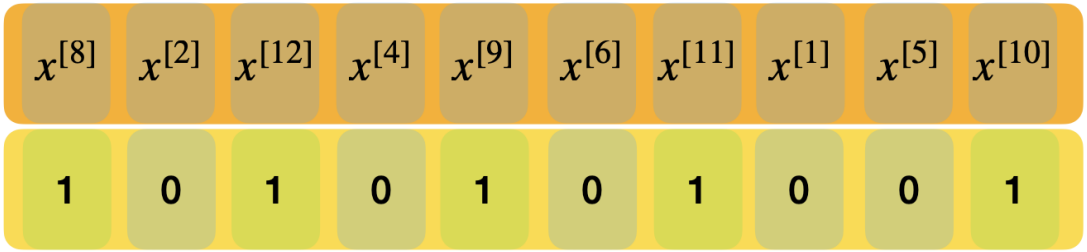
Sử dụng **test set** để đánh giá mô hình



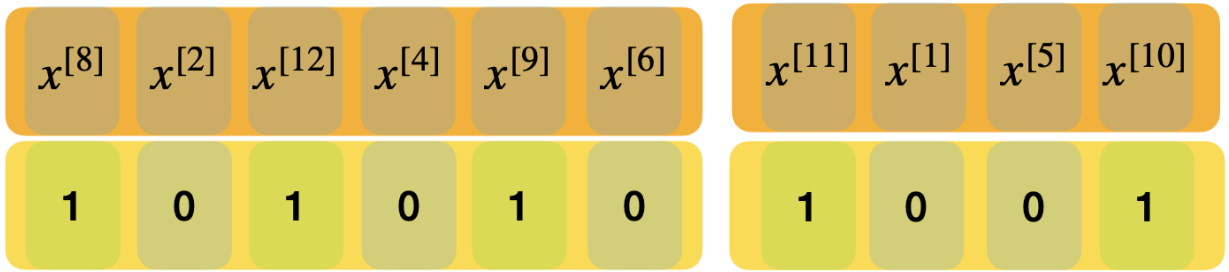
Hình 3: We can use the separate test data set to evaluate the performance of our trained model.

Validation set

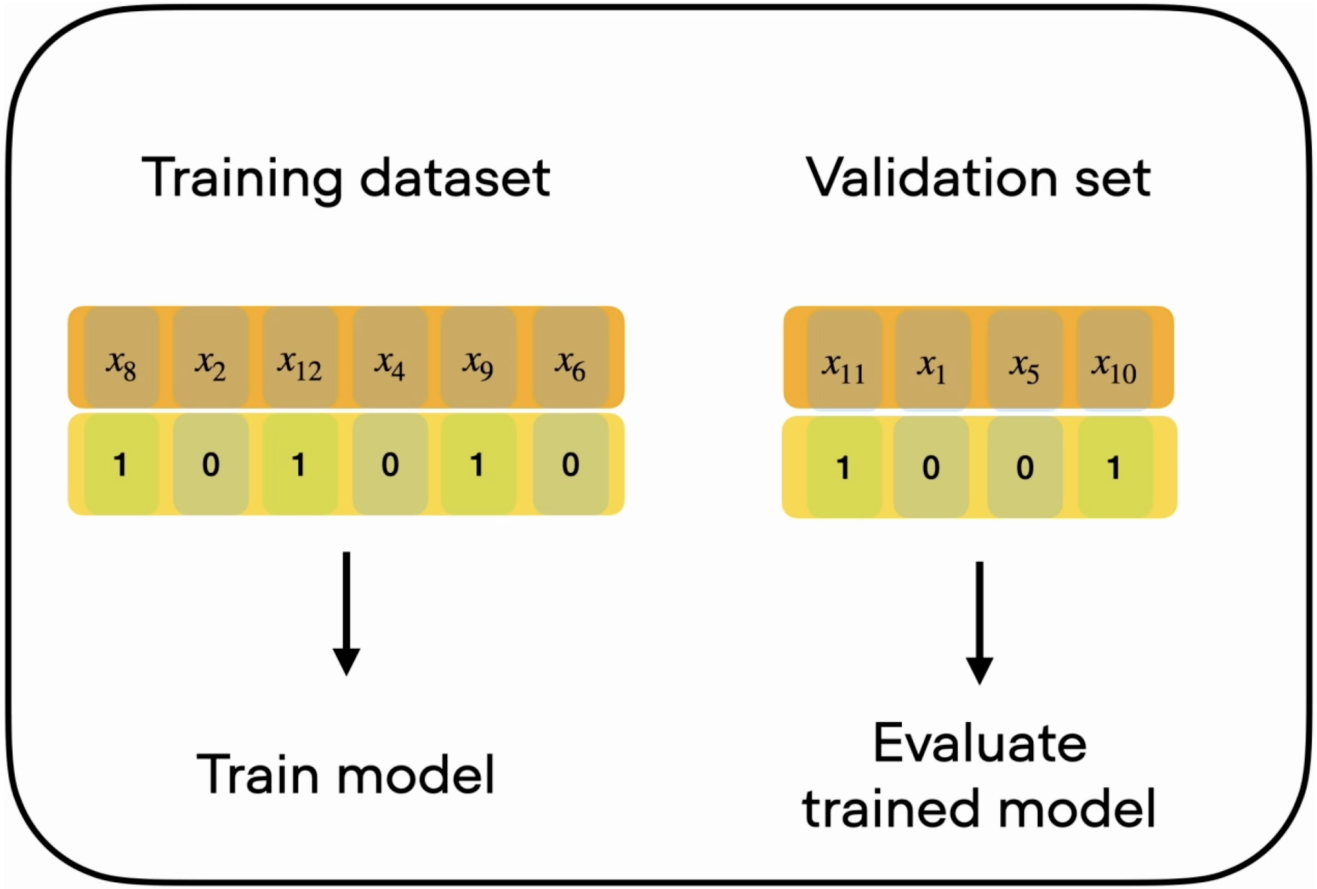
Validation set is an additional split off from the training data set. In other words, we use the training data set to train the model and we use the validation set to tune our model.



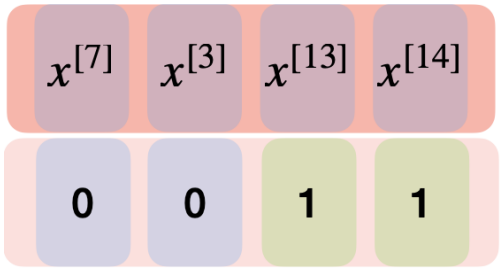
Hình 4: Chia tập training thêm 1 tập Validation để đánh giá hiệu quả huấn luyện.



(a) Training dataset (b) Validation set



(c) Model tuning: repeat this many times

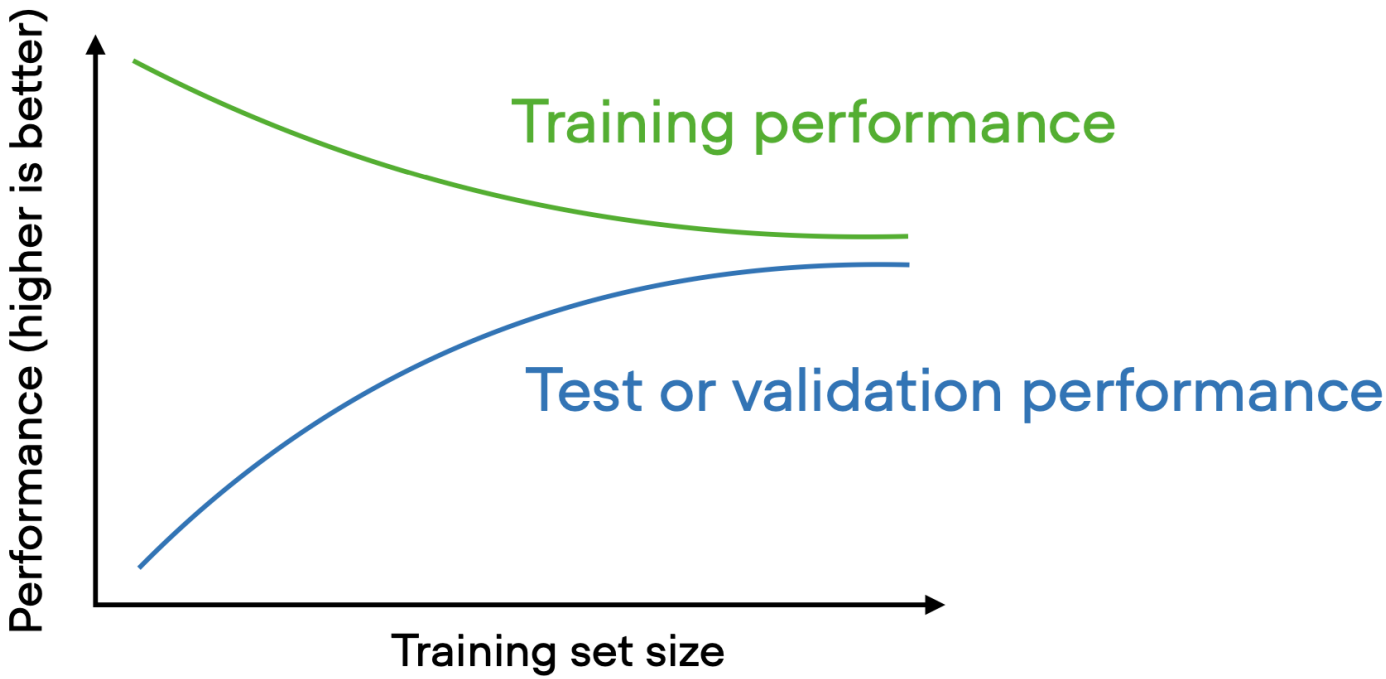


(d) Test dataset

Hình 5: Mục đích của việc chia bộ dữ liệu thành nhiều tập dữ liệu là để đánh giá được tính chính xác hoặc đúng đắn về hiệu suất mô hình. Trong đó, tập test và validation được xem như đại diện cho việc đánh giá trên bộ dữ liệu mới.

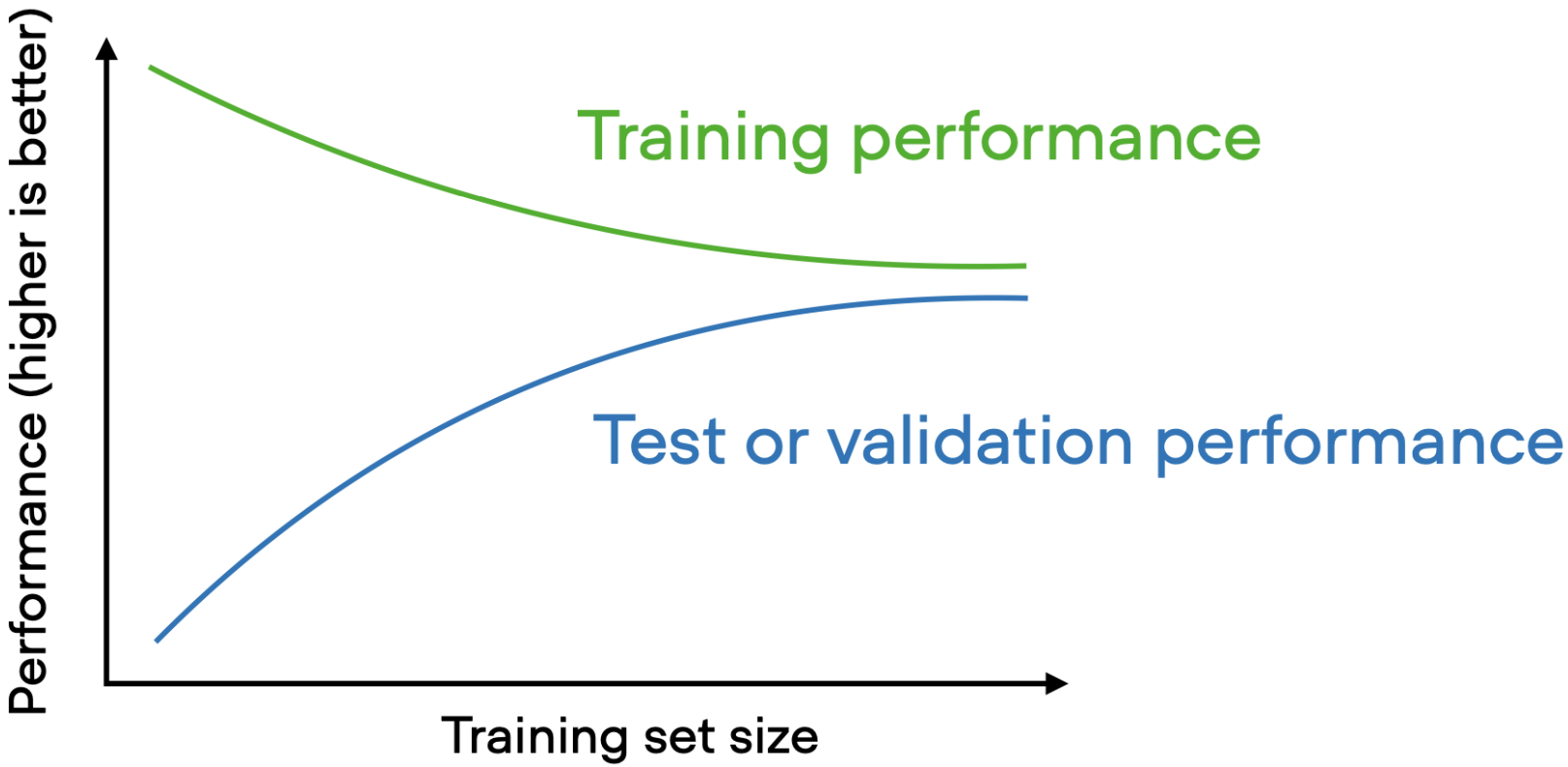
Overfitting

Overfitting means that the model fits the training data too closely. So if we would use the training data set to evaluate the performance of our model, our performance estimate might be actually better than it really is.



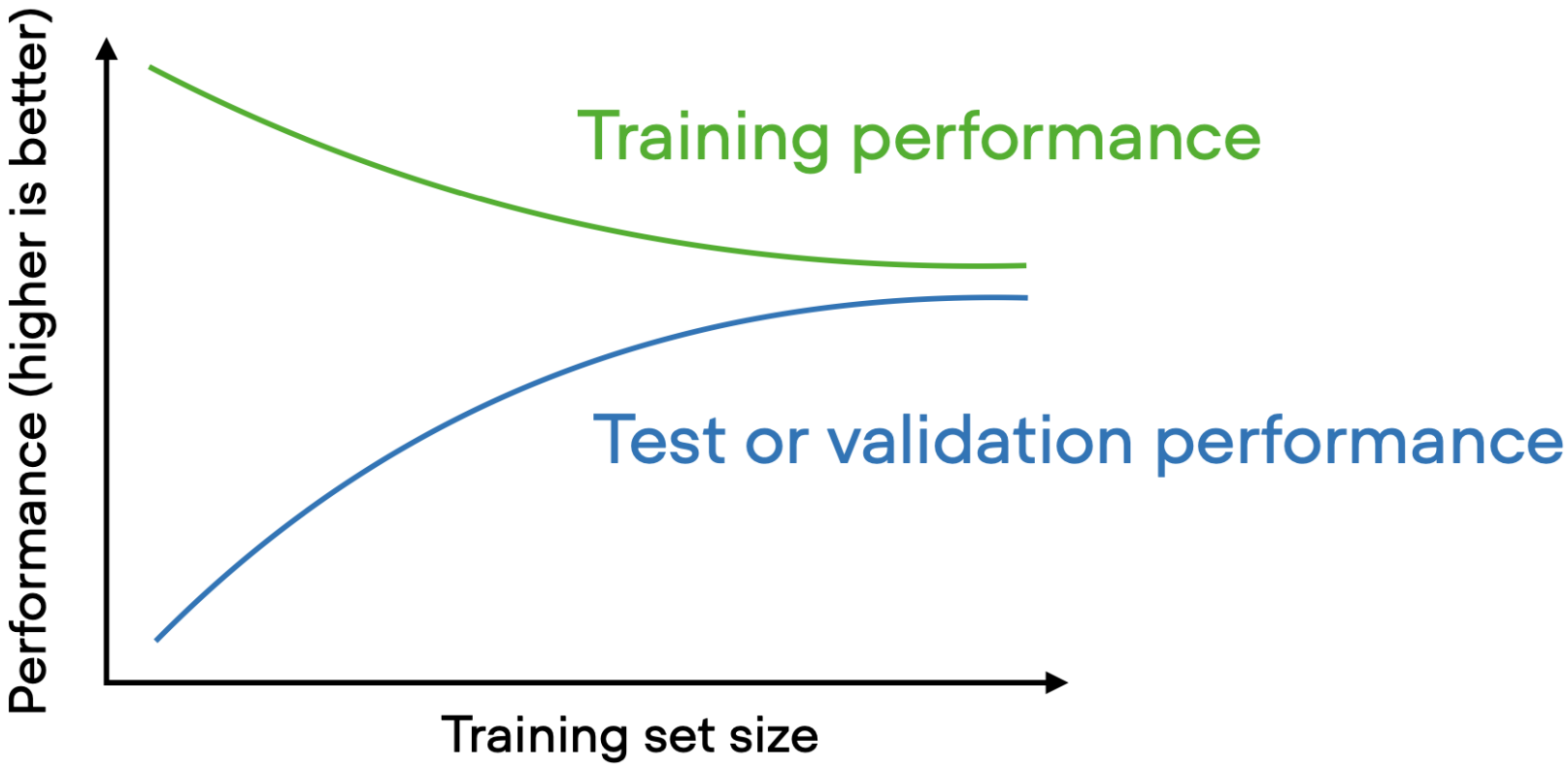
Hình 6: The model performance for different training set sizes.

Small training sets are easy to memorize by deep neural networks.

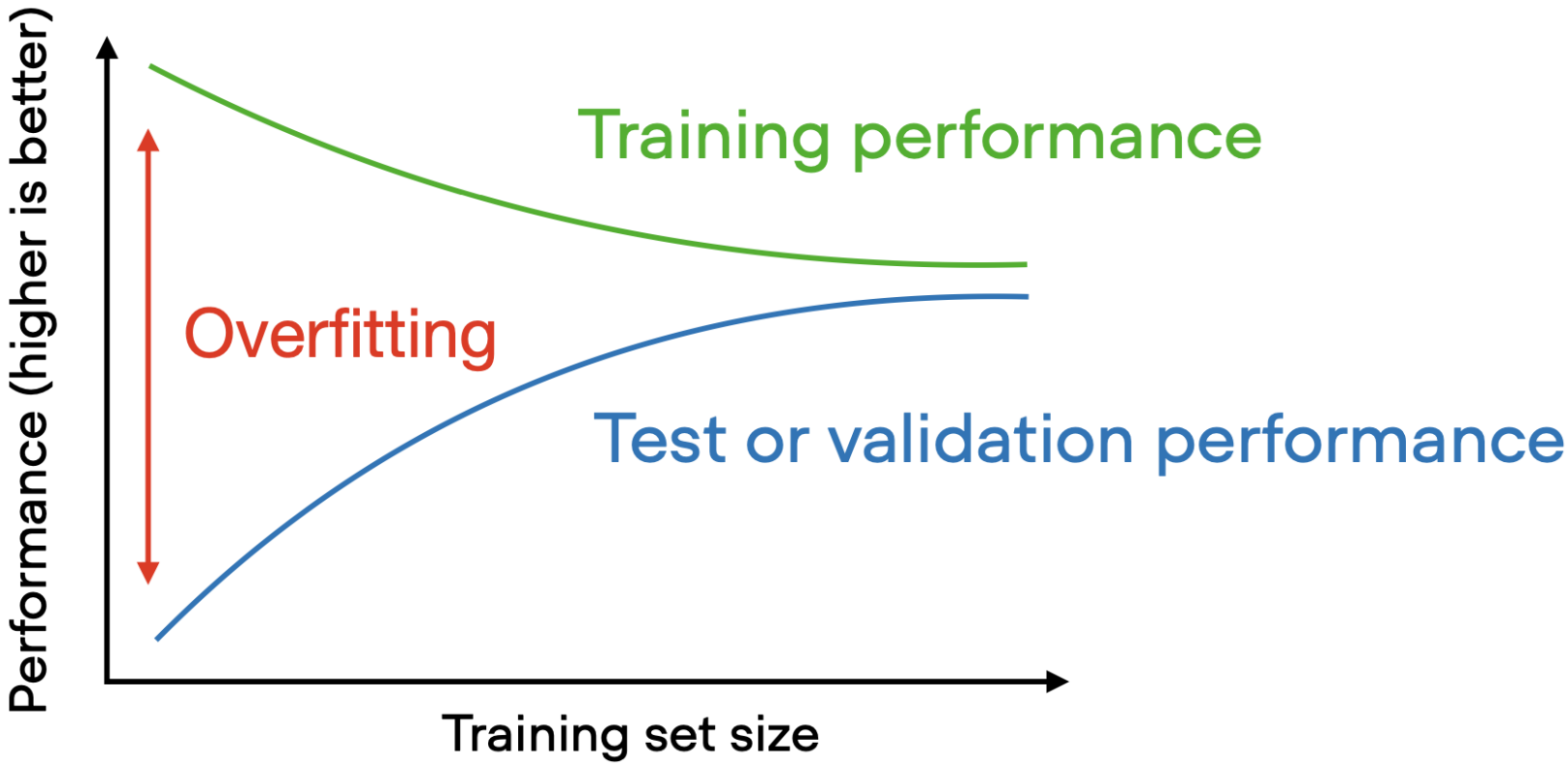


Hình 7: Often when we have a very small training data set, we will get a very high training performance or high training accuracy.

Thực tế, chúng ta thường quan tâm đến hiệu suất ước lượng mô hình trên tập dữ liệu mới thông qua tập validation và test. Rõ ràng, huấn luyện trên tập training lớn sẽ khó khớp (fitting) dữ liệu hơn tập dữ liệu nhỏ.



Hình 8: The larger the data set, typically the easier it is to train a model without overfitting



Hình 9: The effect of overfitting, which is really the gap between training and test set performance.

And typically for machine learning and especially deep learning models, the larger the data set, the better. While the training set performance may go down.

Tài liệu tham khảo

Sebastian Raschka, Yuxi (Hayden) Liu, Vahid Mirjalili
Machine Learning with PyTorch and Scikit-Learn: Develop machine learning and deep learning models with Python (2022). Published by Packt Publishing Ltd, ISBN 978-1-80181-931-2.

Sebastian Raschka
MACHINE LEARNING Q AND AI: 30 Essential Questions and Answers on Machine Learning and AI (2024). ISBN-13: 978-1-7185-0377-9 (ebook).

LightningAI
LightningAI: PyTorch Lightning (2024) .