The Perceptron

LING 282/482: Deep Learning for Computational Linguistics

C.M. Downey

Fall 2025

Supervised Learning Basics

Supervised Learning Basics

- Overall idea: learn a **mapping** between **inputs** X and **outputs** Y

Supervised Learning Basics

- Overall idea: learn a **mapping** between **inputs** X and **outputs** Y
  - In math terms, learning a **function** $f(x) = y$

Supervised Learning Basics

- Overall idea: learn a **mapping** between **inputs** X and **outputs** Y
  - In math terms, learning a **function** $f(x) = y$
- The function is learned from a **dataset** of examples

Supervised Learning Basics

- Overall idea: learn a **mapping** between **inputs** X and **outputs** Y
  - In math terms, learning a **function** $f(x) = y$
- The function is learned from a **dataset** of examples
  - $D = \{(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)\}$

Supervised Learning Basics

- Overall idea: learn a **mapping** between **inputs** X and **outputs** Y
  - In math terms, learning a **function** $f(x) = y$
- The function is learned from a **dataset** of examples
  - $D = \{(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)\}$
  - The dataset contains **pairs of inputs and outputs**

Supervised Learning Basics

- Overall idea: learn a **mapping** between **inputs** X and **outputs** Y
  - In math terms, learning a **function** $f(x) = y$
- The function is learned from a **dataset** of examples
  - $D = \{(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)\}$
  - The dataset contains **pairs of inputs and outputs**
  - Ex: speed $\rightarrow$ whether you get a speeding ticket

Supervised Learning Basics

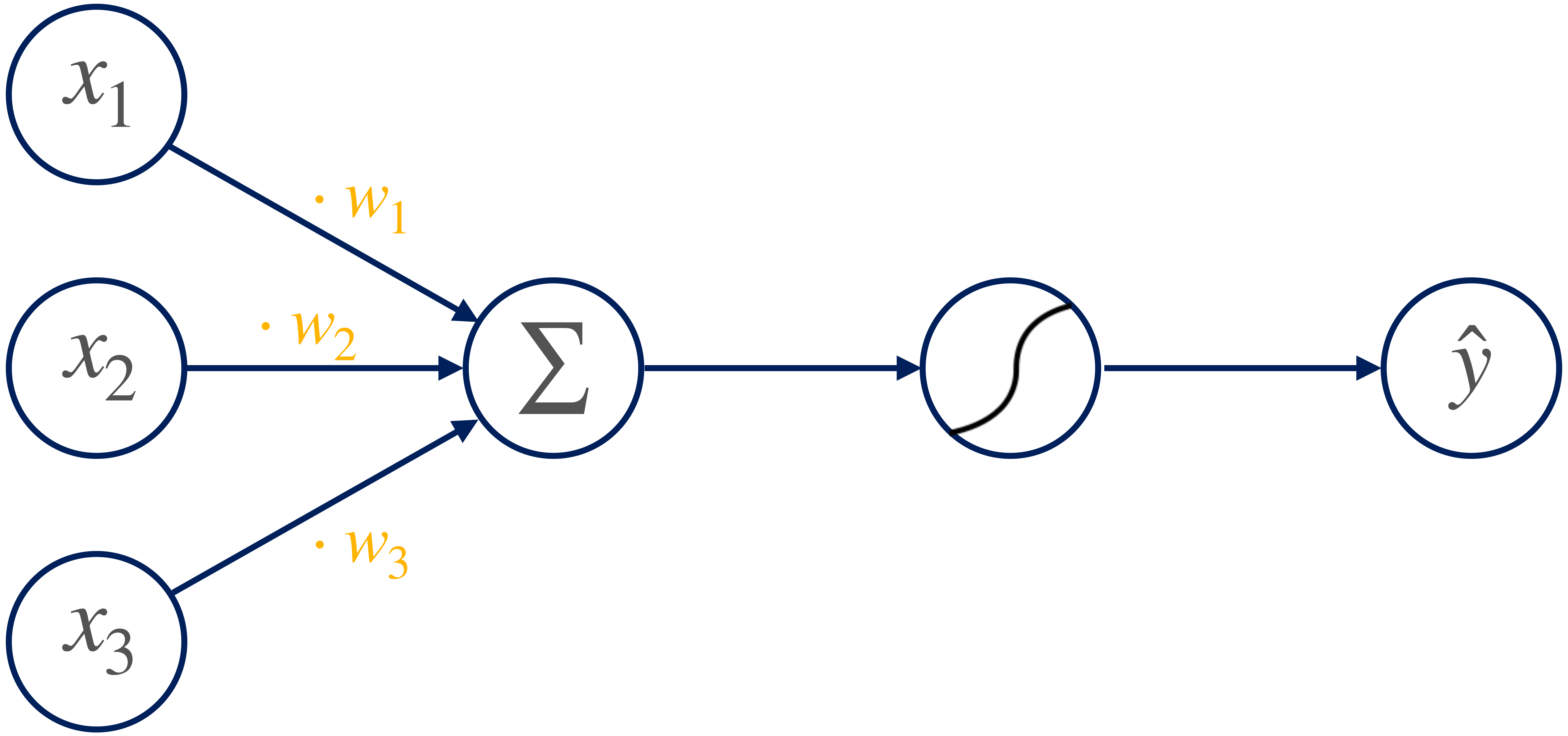
- Overall idea: learn a **mapping** between **inputs** X and **outputs** Y
  - In math terms, learning a **function** $f(x) = y$
- The function is learned from a **dataset** of examples
  - $D = \{(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)\}$
  - The dataset contains **pairs of inputs and outputs**
  - Ex: speed $\rightarrow$ whether you get a speeding ticket
    - {(30, False), (33, False), (35, False), (37, True), (39, True)}

Supervised Learning Basics

- Overall idea: learn a **mapping** between **inputs** X and **outputs** Y
  - In math terms, learning a **function** $f(x) = y$
- The function is learned from a **dataset** of examples
  - $D = \{(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)\}$
  - The dataset contains **pairs of inputs and outputs**
  - Ex: speed $\rightarrow$ whether you get a speeding ticket
    - {(30, False), (33, False), (35, False), (37, True), (39, True)}
- Goal: learn the function that **best matches the dataset**

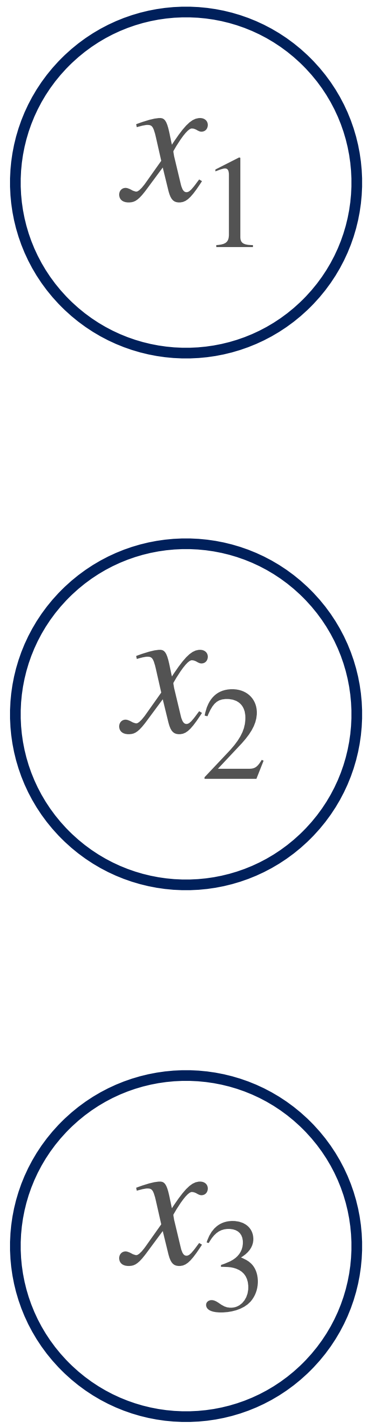
Perceptron

- Last lecture: **vectors, matrices, and linear transformations**
  - How do these relate to Neural Networks?
- We'll demonstrate using the **simplest Neural Network**: the **Perceptron**



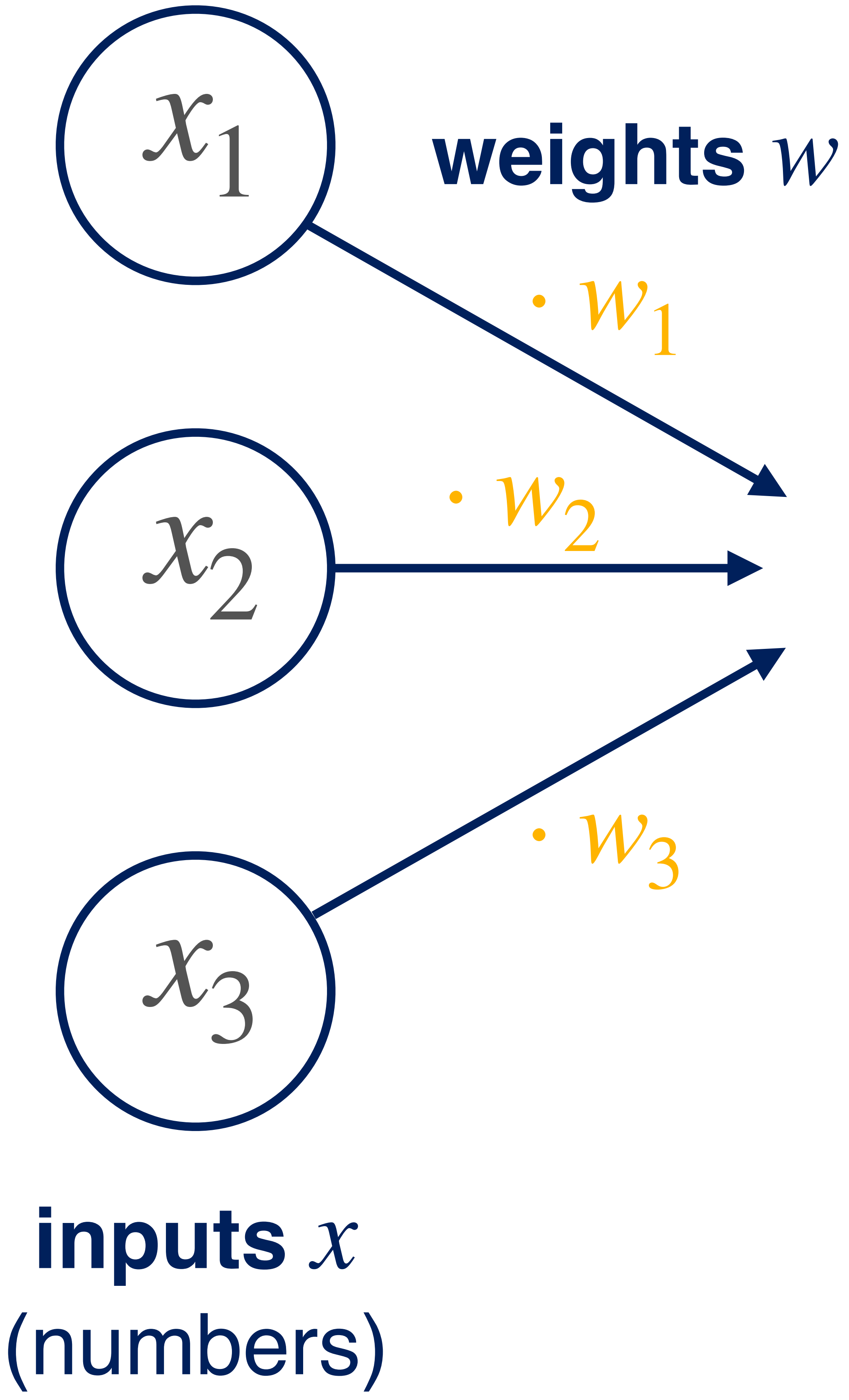
Perceptron Components

Perceptron Components

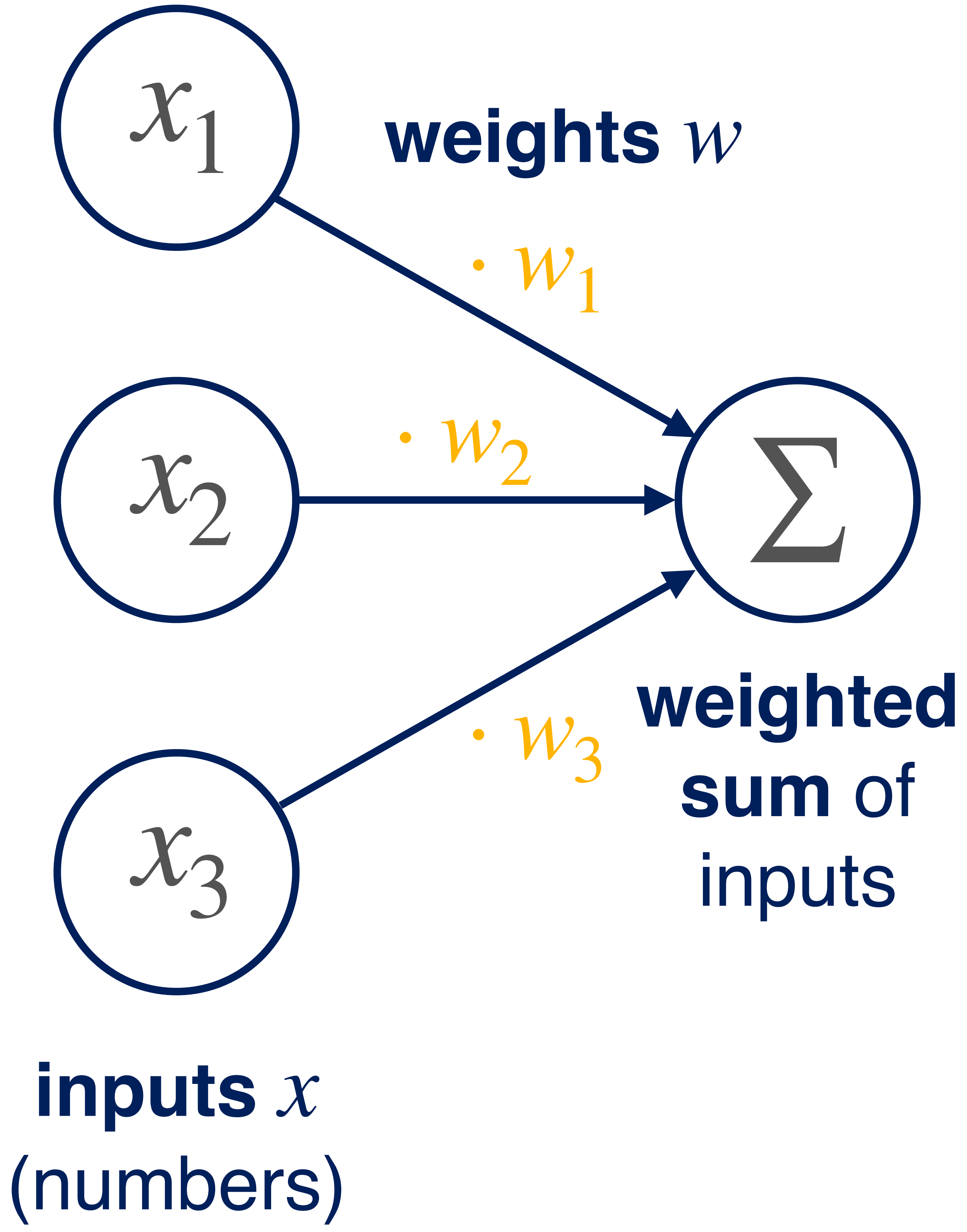


inputs x
(numbers)

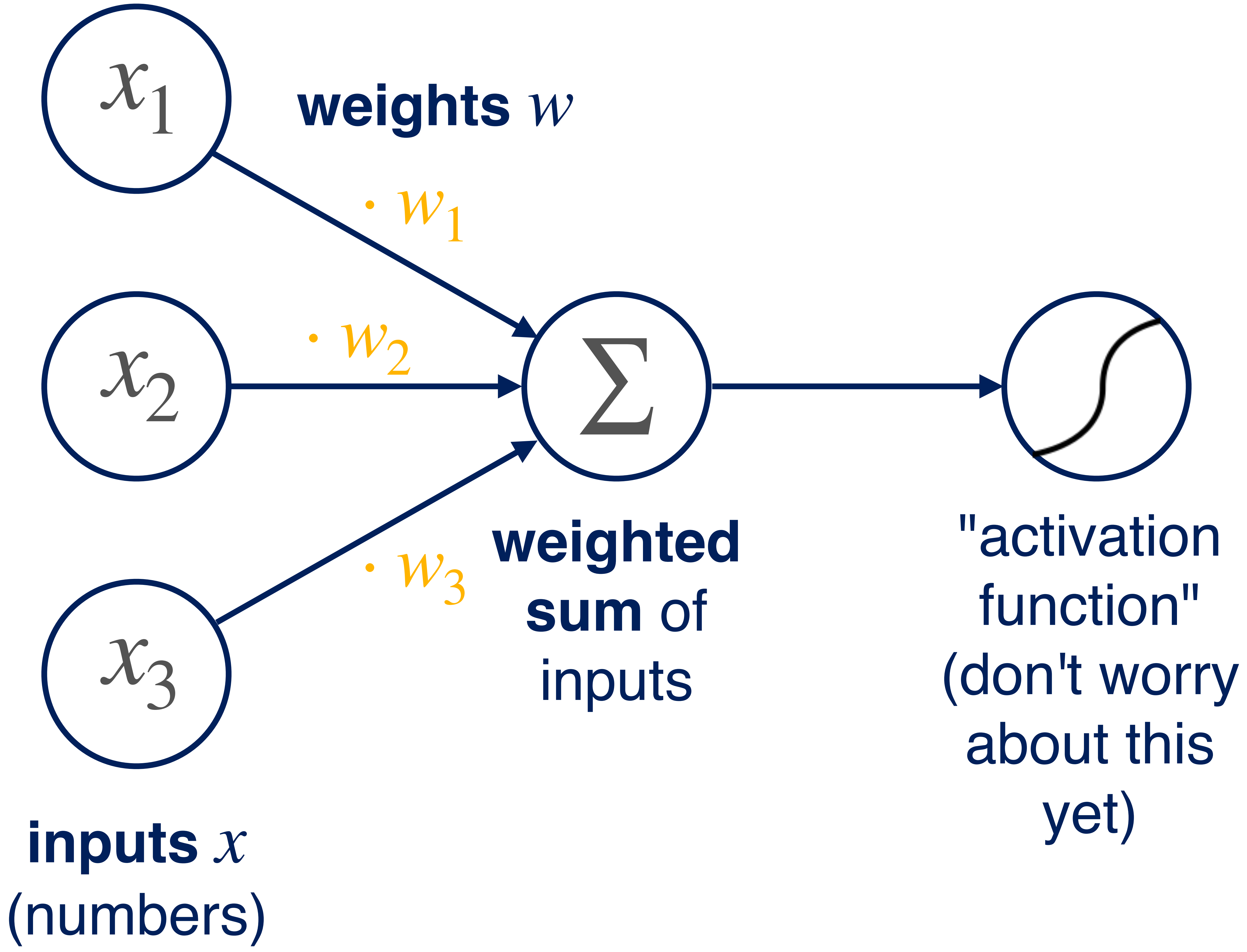
Perceptron Components



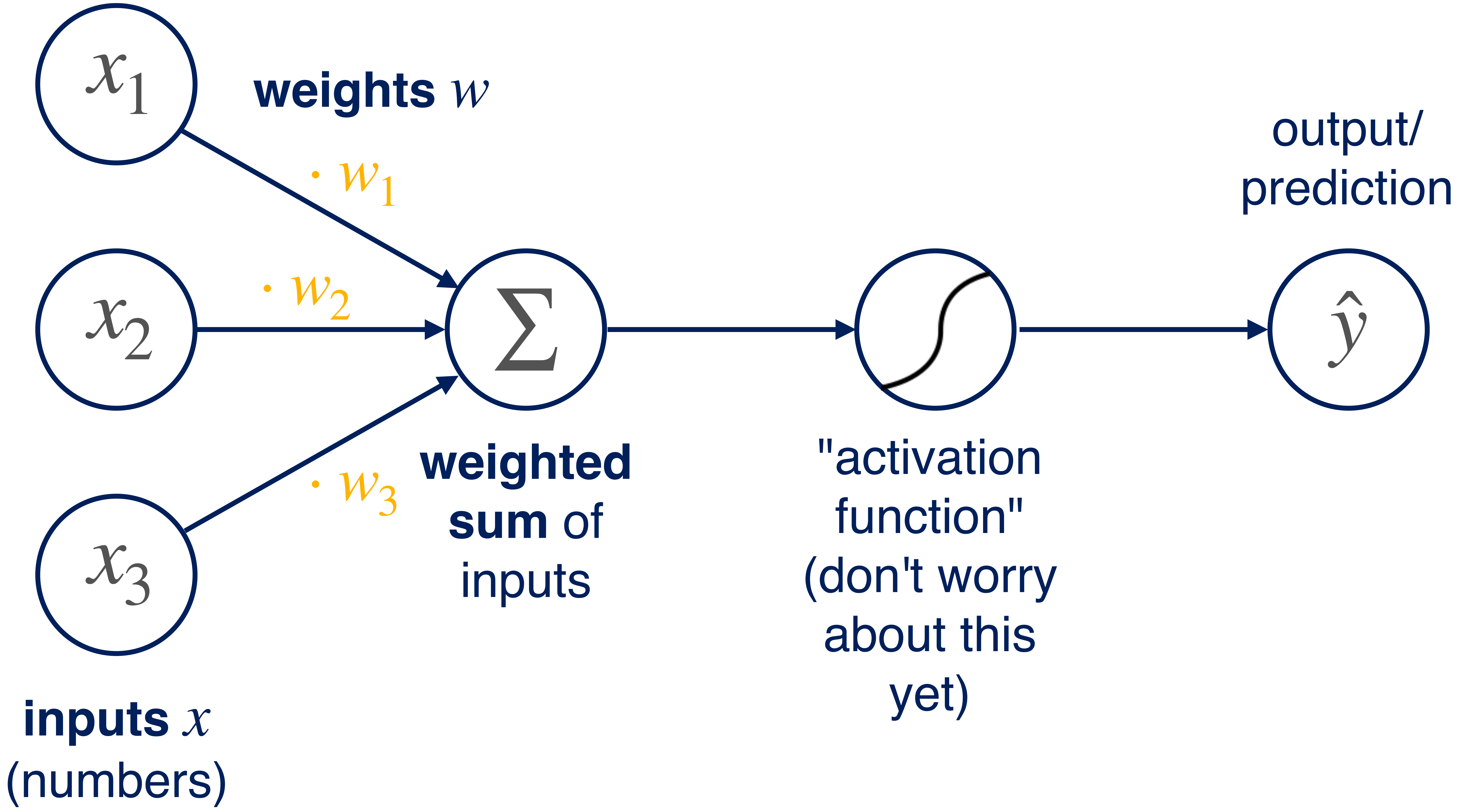
Perceptron Components



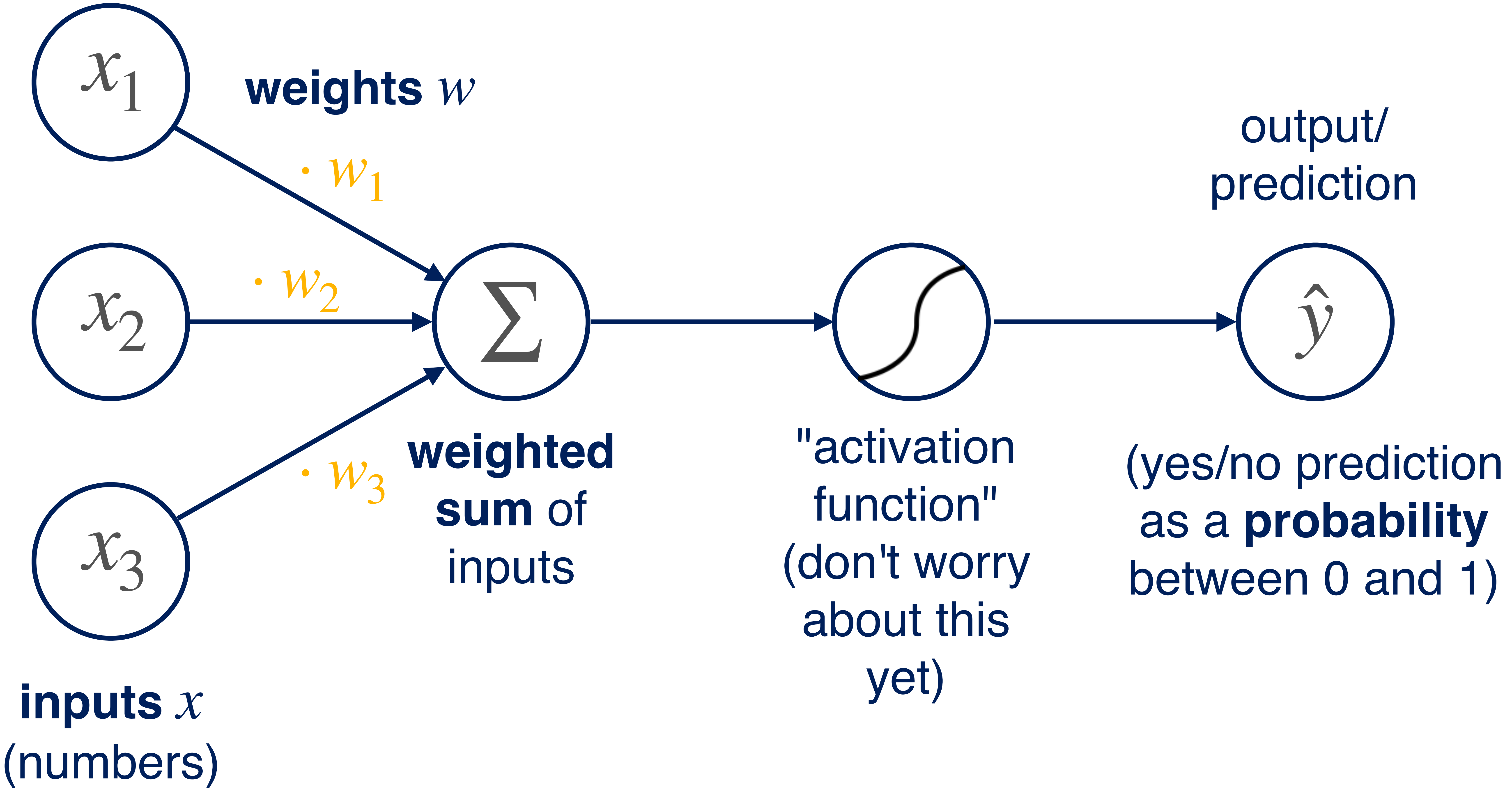
Perceptron Components



Perceptron Components

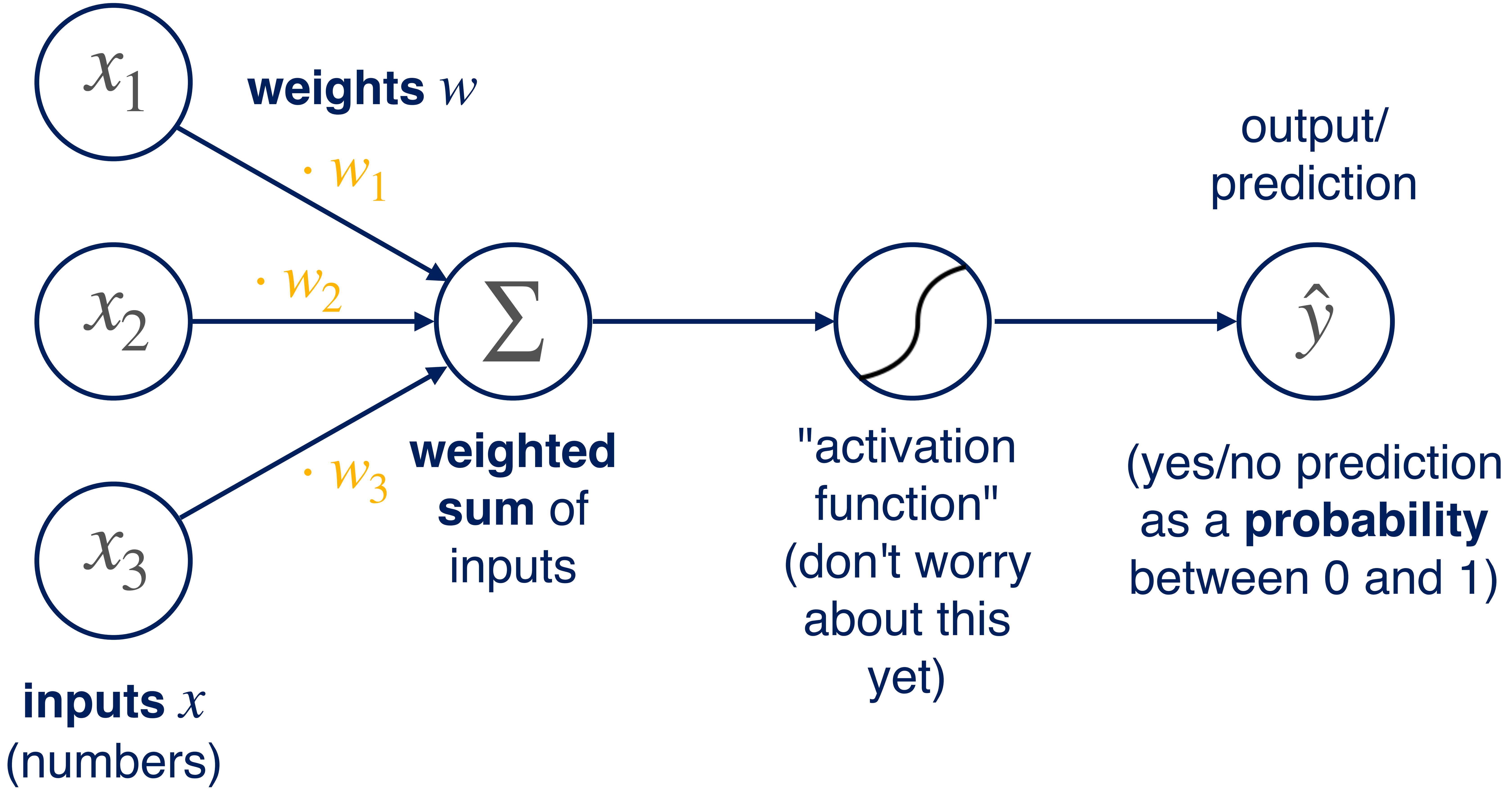


Perceptron Components



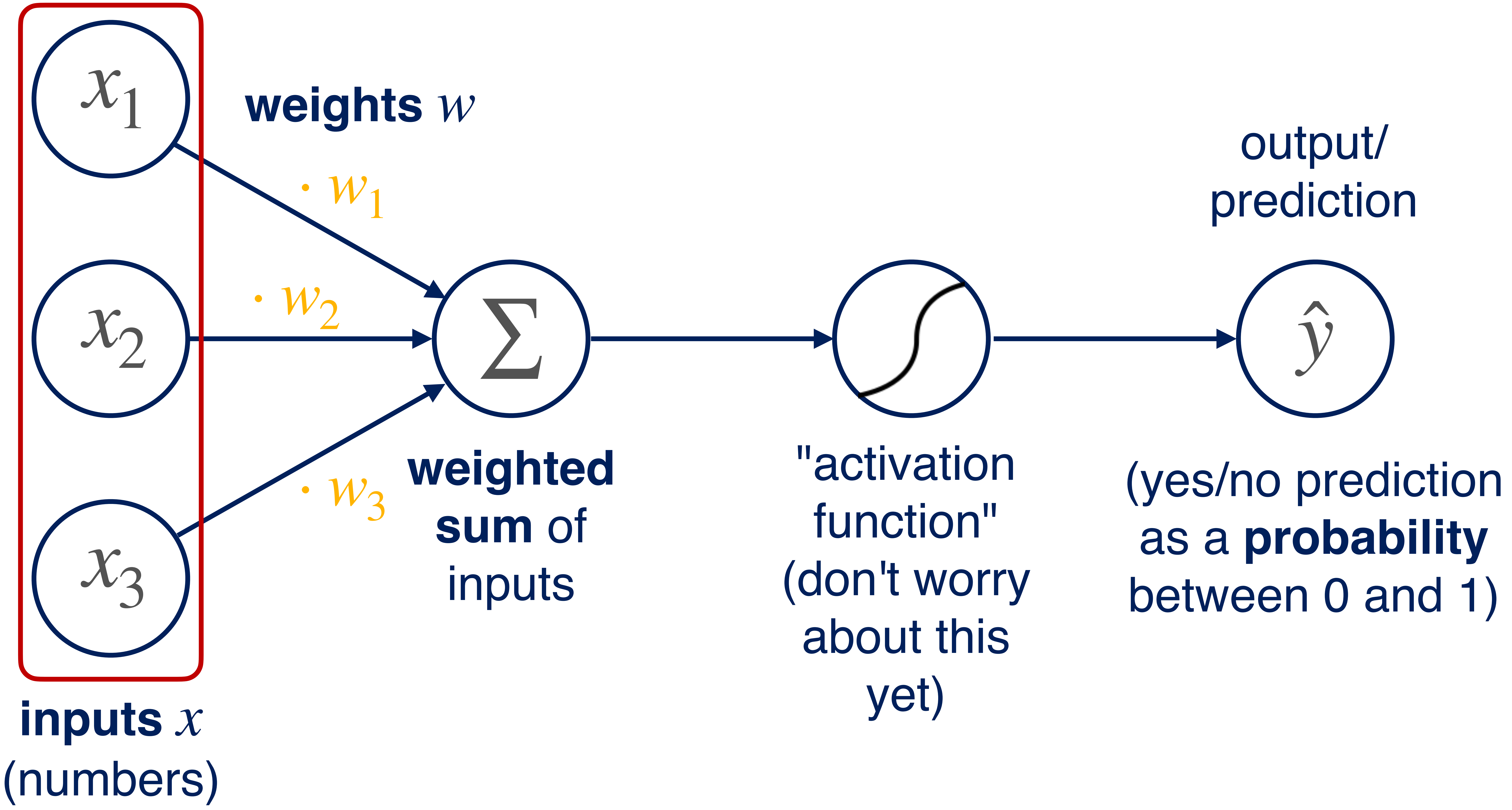
Perceptron Components

Where are the vectors?



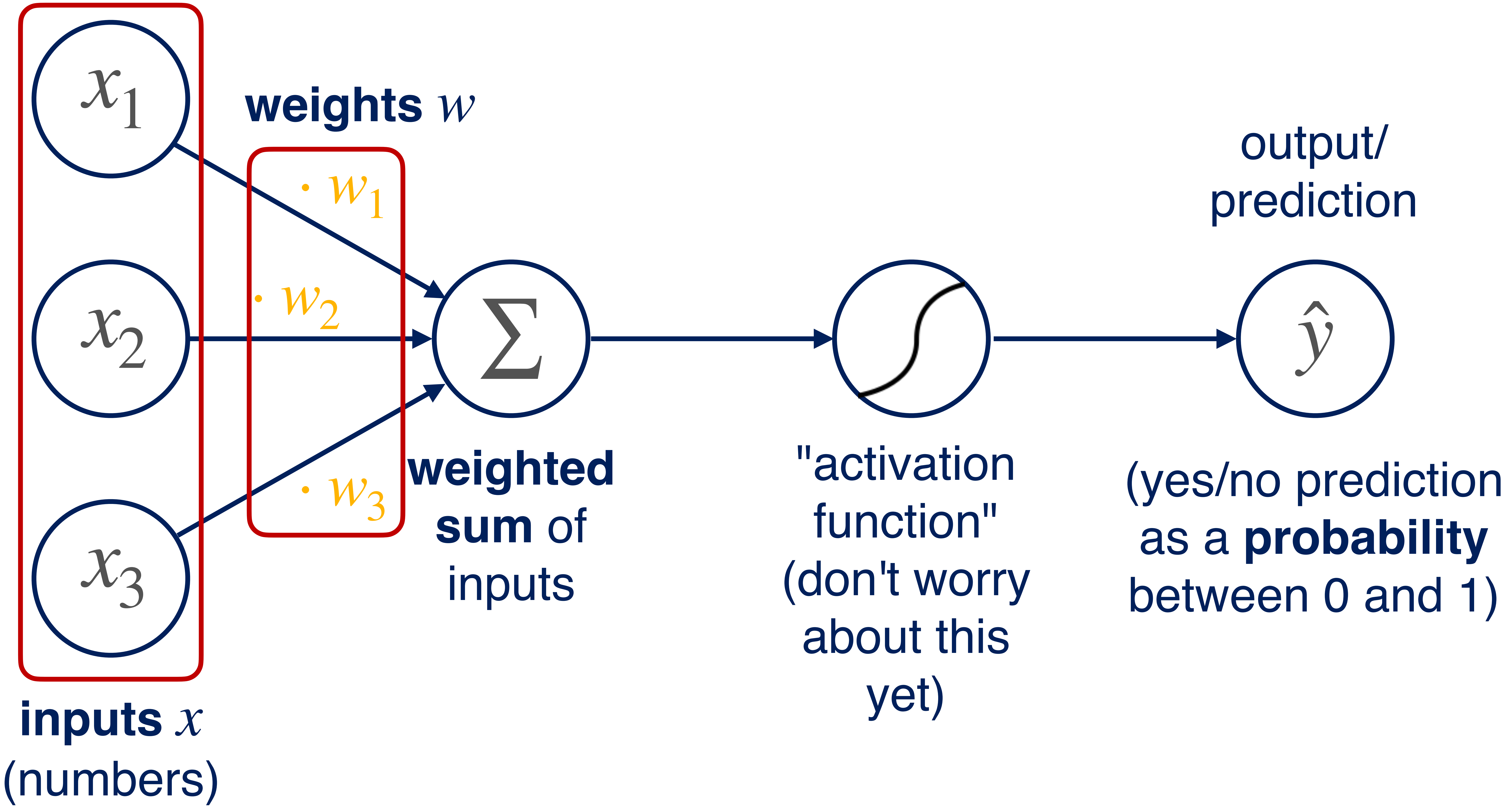
Perceptron Components

Where are the vectors?

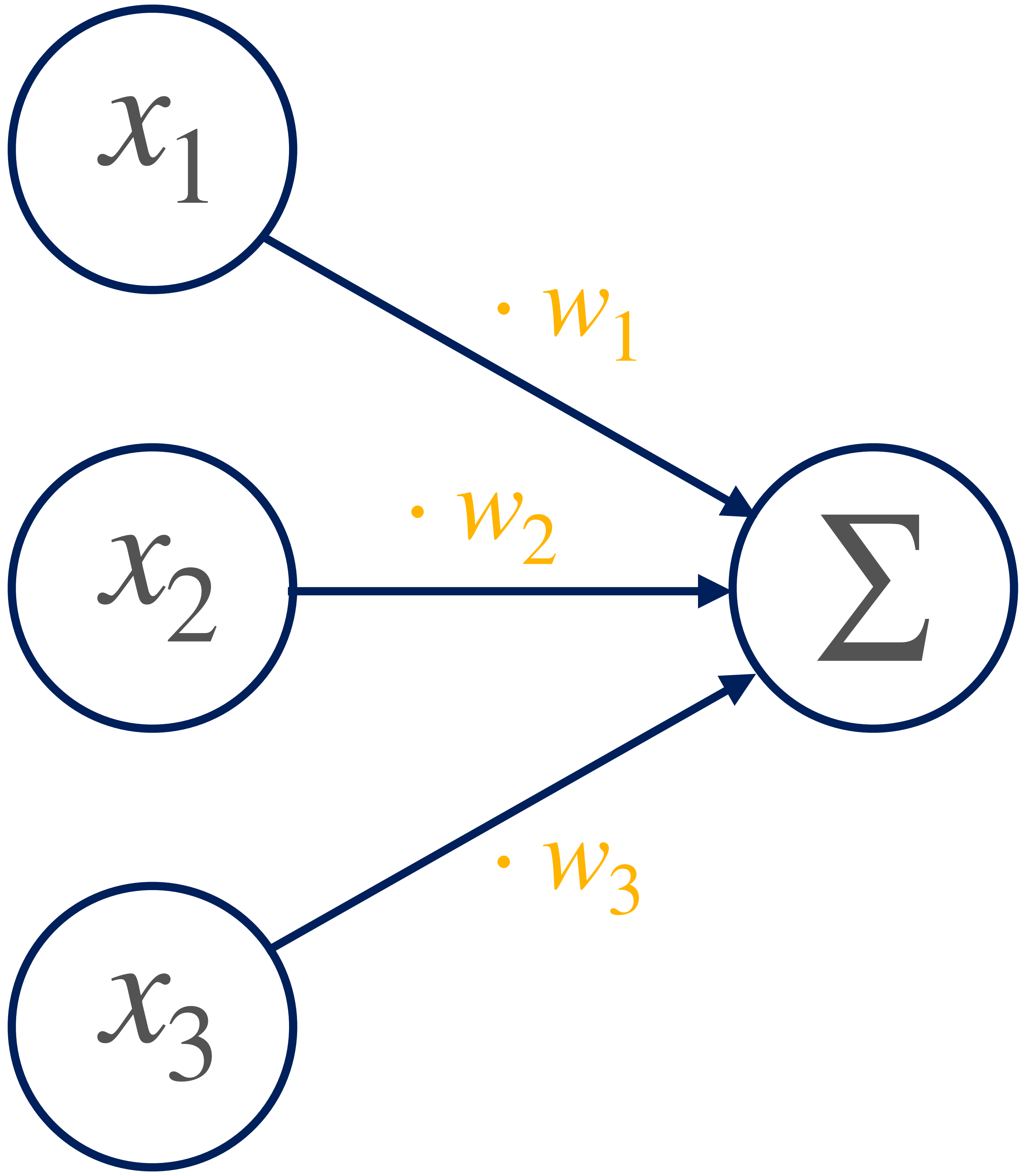


Perceptron Components

Where are the vectors?

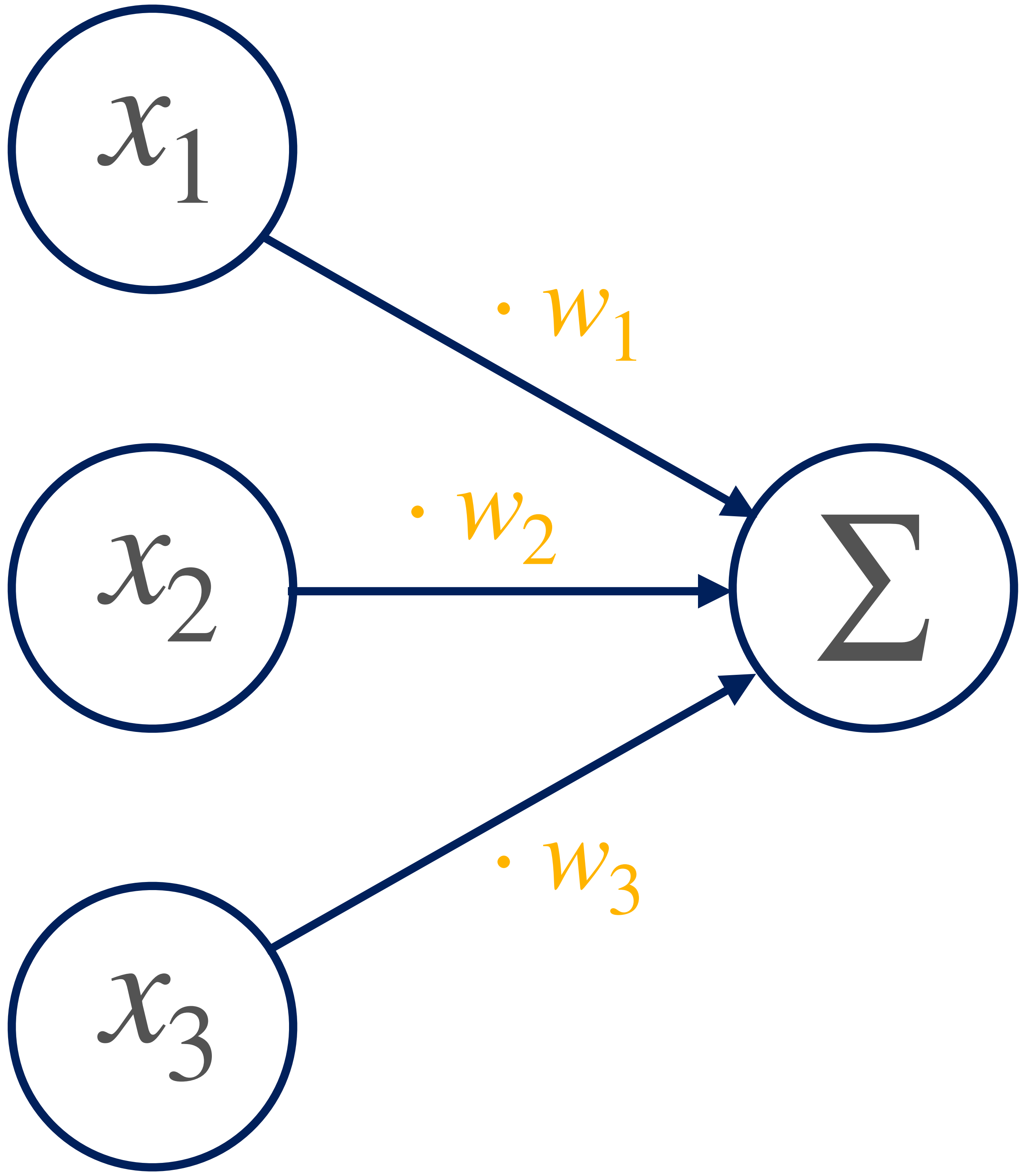


Vectors in the Perceptron



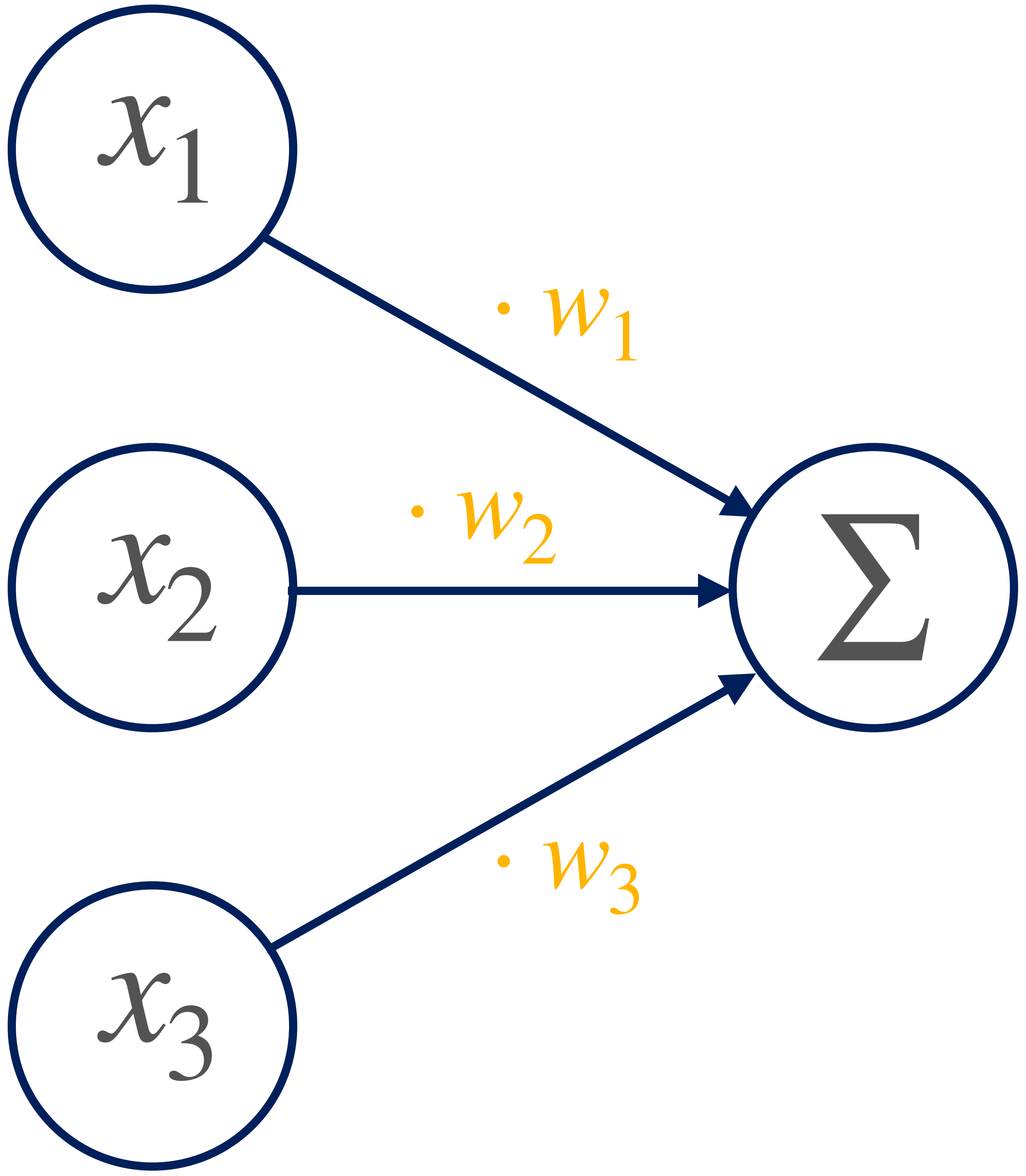
Vectors in the Perceptron

- The **weighted sum** of the perceptron...



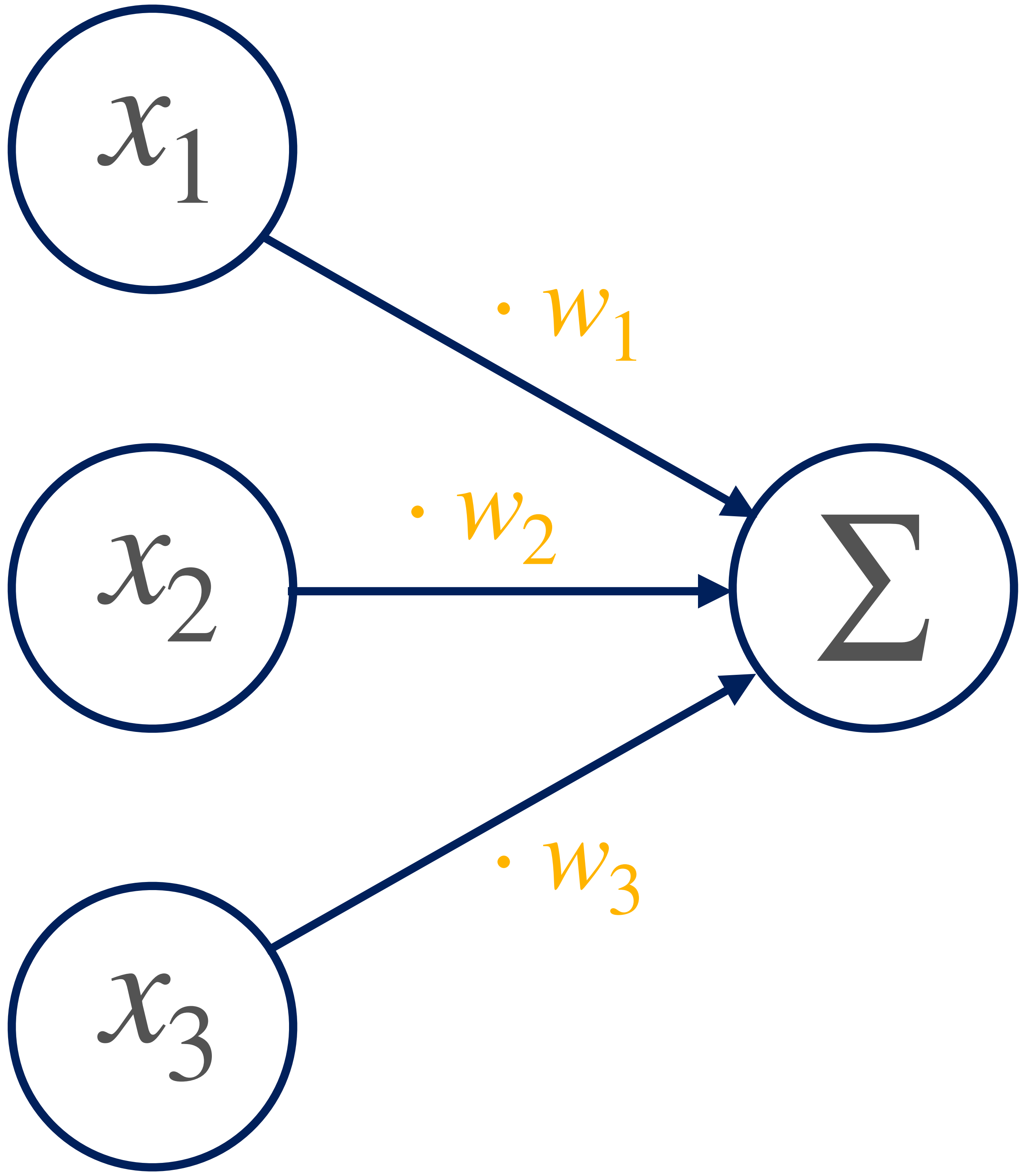
Vectors in the Perceptron

- The **weighted sum** of the perceptron...
  - takes each element of the **input vector** x...



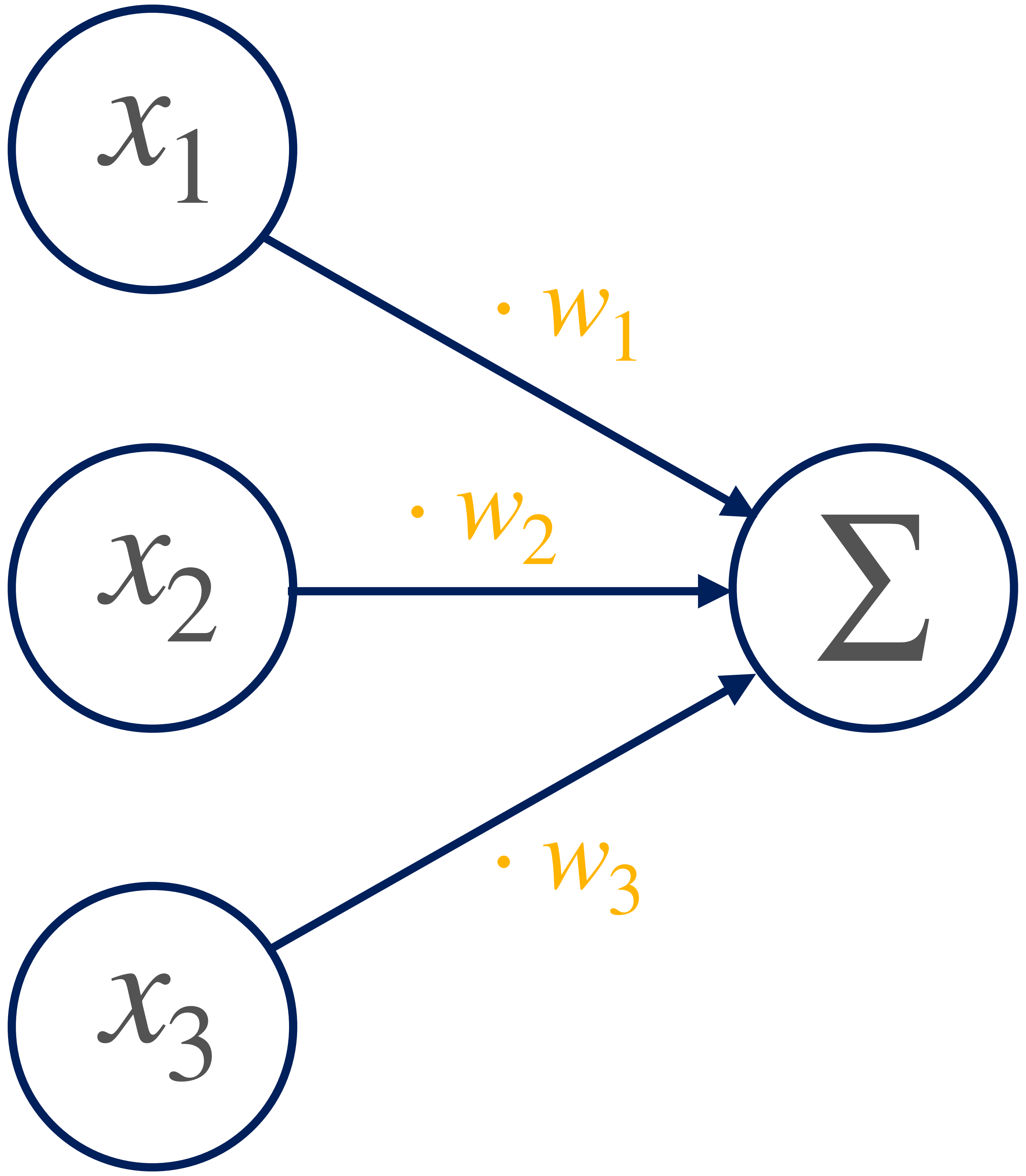
Vectors in the Perceptron

- The **weighted sum** of the perceptron...
  - takes each element of the **input vector** x...
  - times the element of the **weight vector** w:



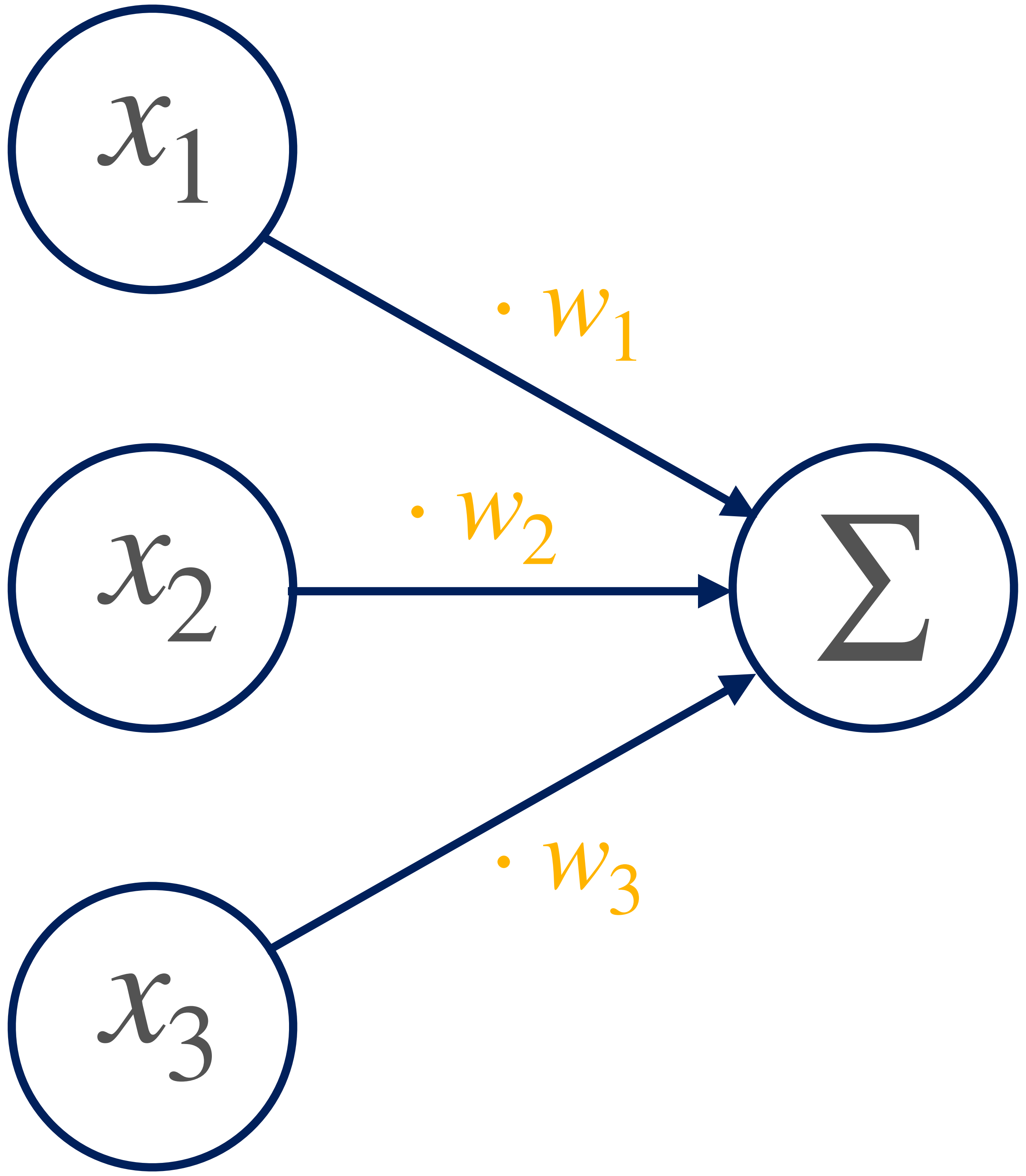
Vectors in the Perceptron

- The **weighted sum** of the perceptron...
  - takes each element of the **input vector** x...
  - times the element of the **weight vector** w:
  - $w_1x_1 + w_2x_2 + \ldots w_nx_n$



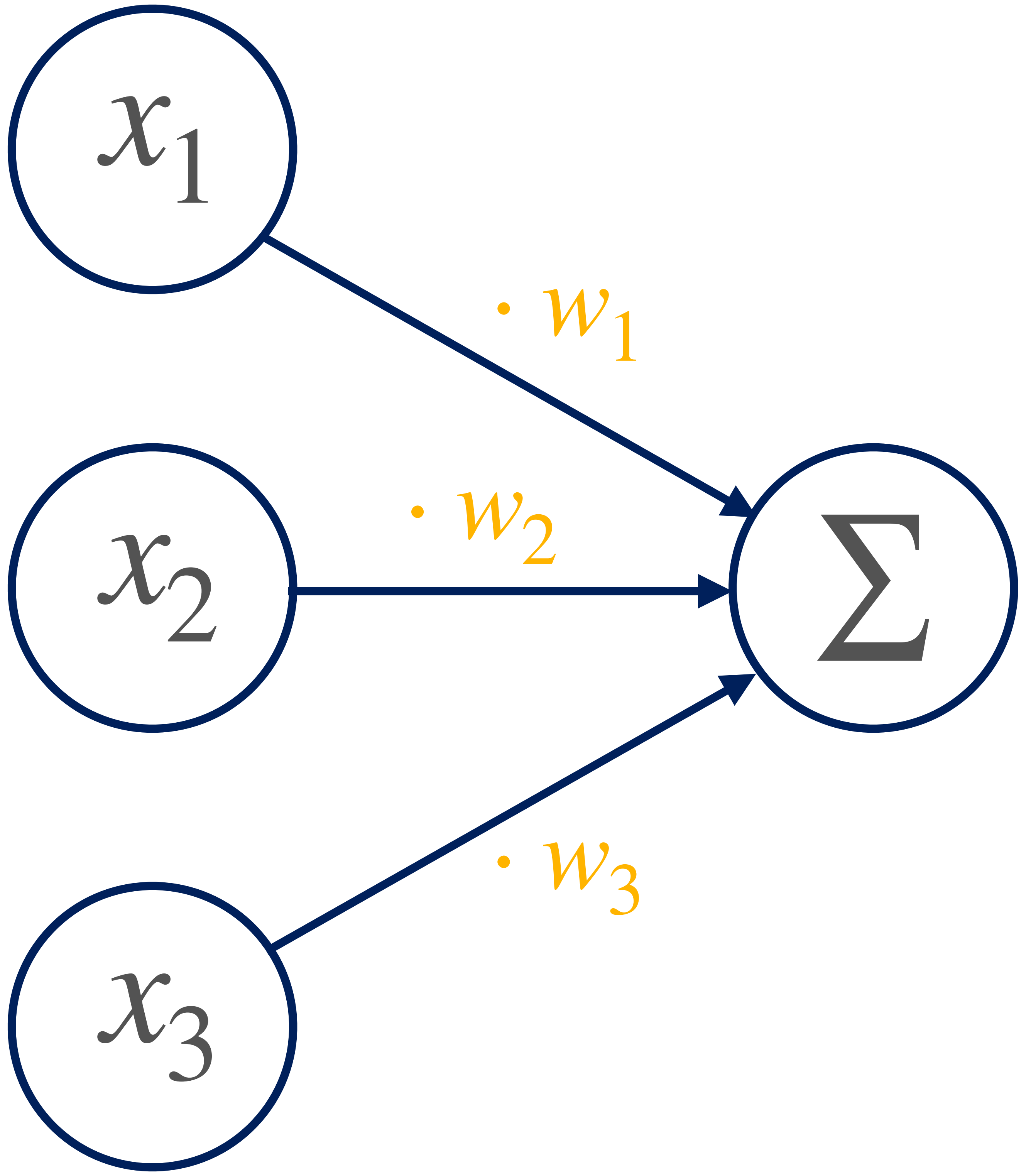
Vectors in the Perceptron

- The **weighted sum** of the perceptron...
  - takes each element of the **input vector** x...
  - times the element of the **weight vector** w:
  - $w_1x_1 + w_2x_2 + \ldots w_nx_n$
  - Where have we **seen this formula before?**



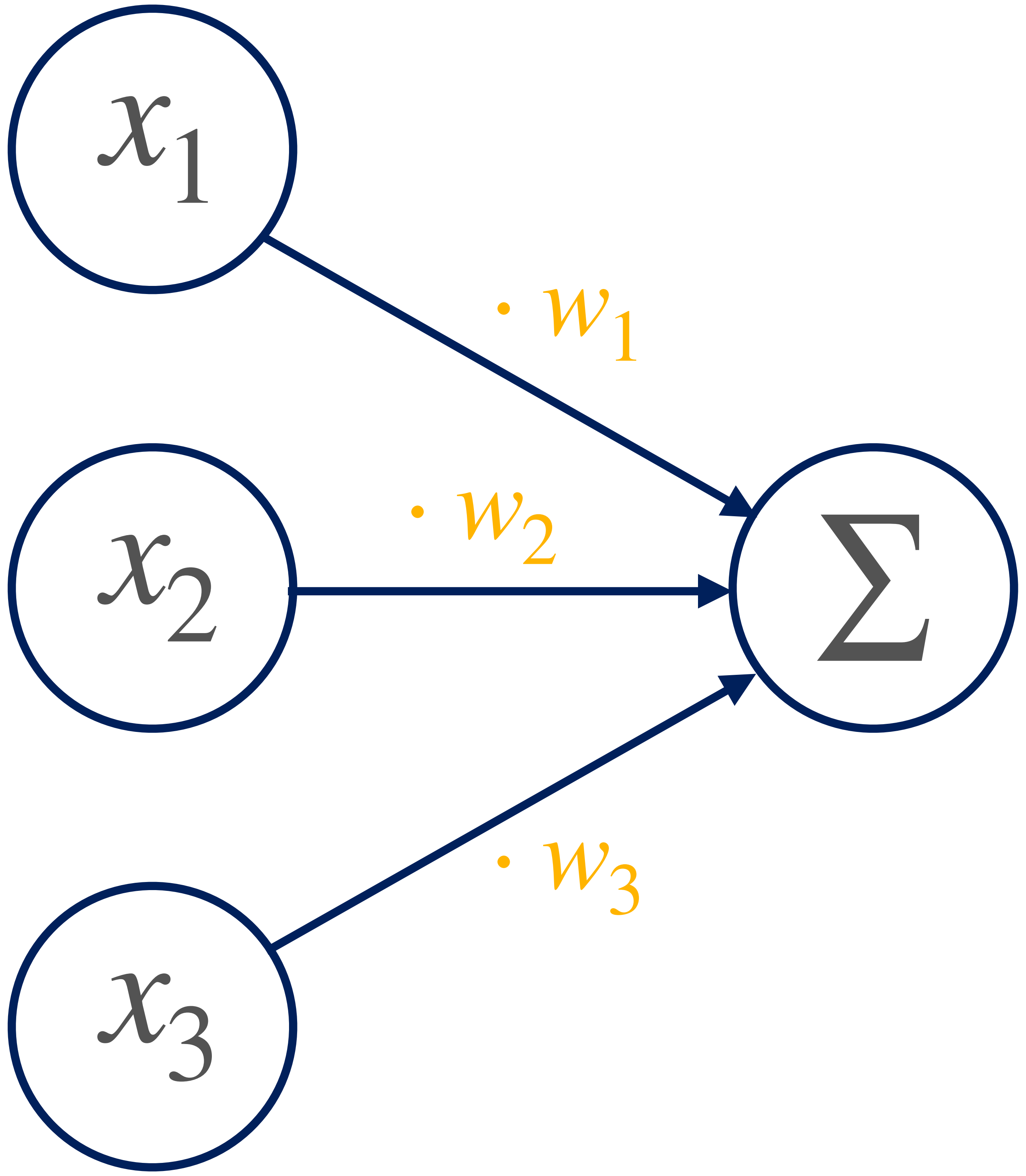
Vectors in the Perceptron

- The **weighted sum** of the perceptron...
  - takes each element of the **input vector** x...
  - times the element of the **weight vector** w:
  - $w_1x_1 + w_2x_2 + \ldots w_nx_n$
  - Where have we **seen this formula before?**
- This is the formula for the **dot product** of the two vectors!



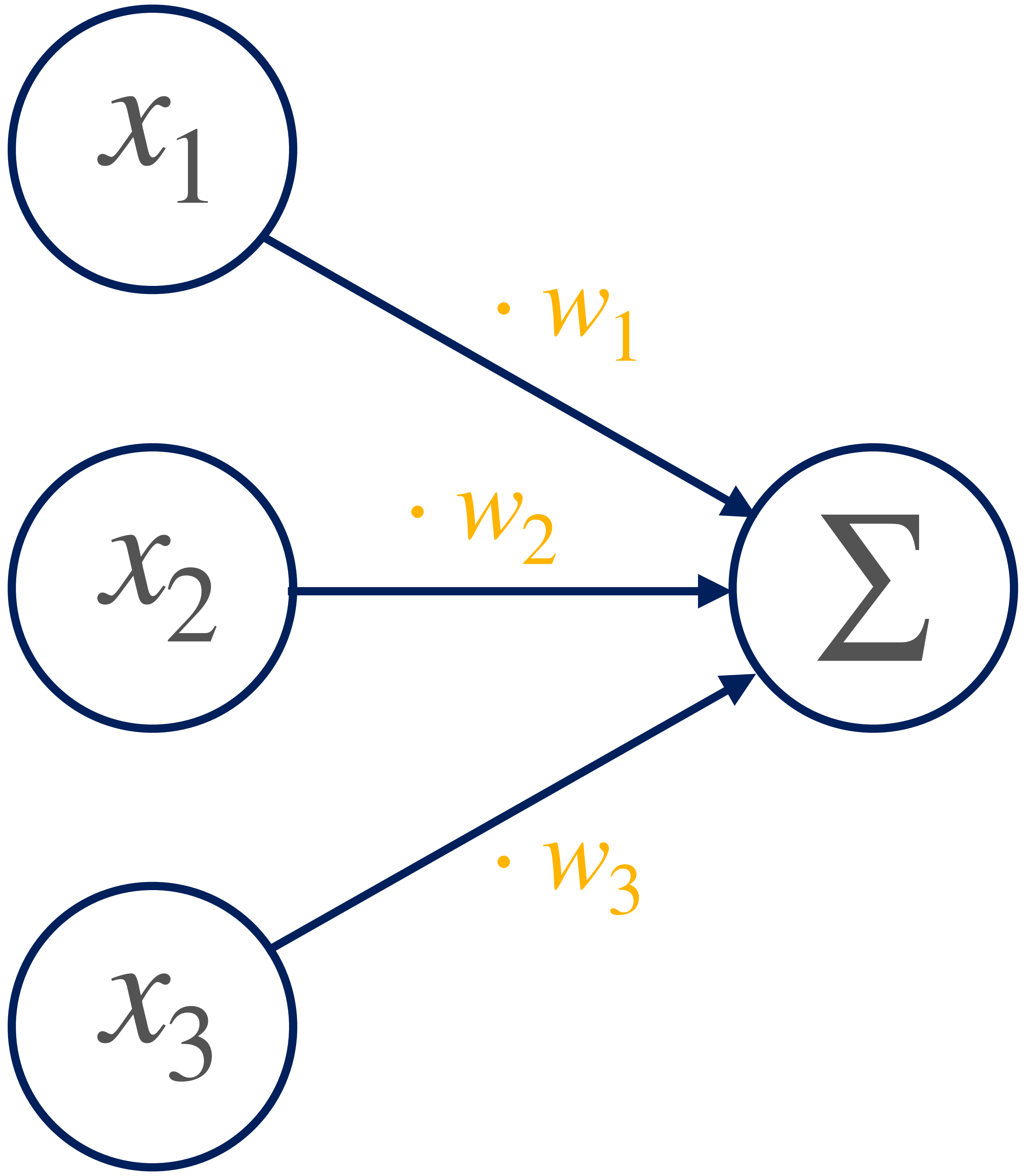
Vectors in the Perceptron

- The **weighted sum** of the perceptron...
  - takes each element of the **input vector** x...
  - times the element of the **weight vector** w:
  - $w_1x_1 + w_2x_2 + \ldots w_nx_n$
  - Where have we **seen this formula before?**
- This is the formula for the **dot product** of the two vectors!
  - $w \cdot x$ or w^Tx



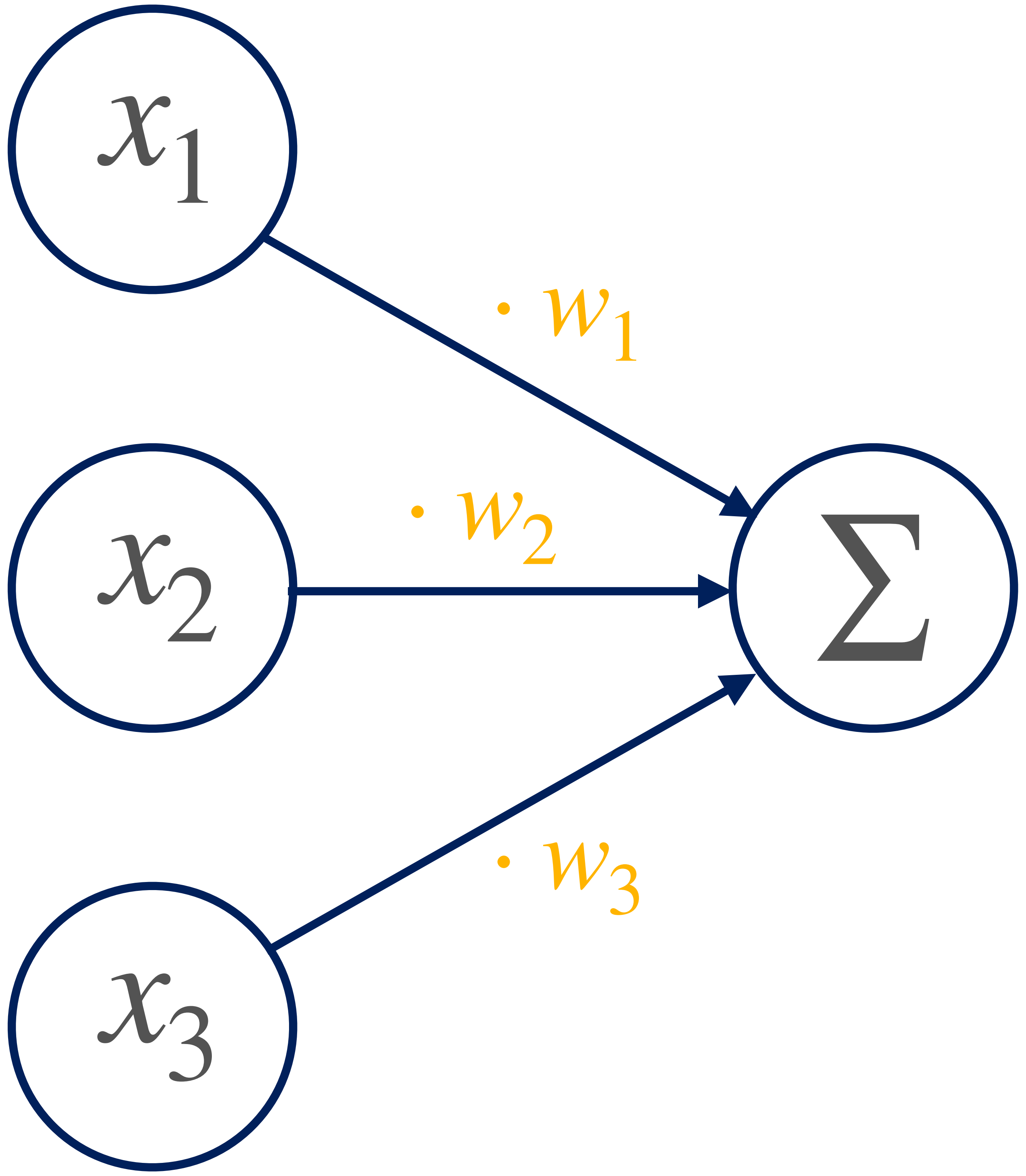
Vectors in the Perceptron

- The **weighted sum** of the perceptron...
  - takes each element of the **input vector** x...
  - times the element of the **weight vector** w:
  - $w_1x_1 + w_2x_2 + \ldots w_nx_n$
  - Where have we **seen this formula before?**
- This is the formula for the **dot product** of the two vectors!
  - $w \cdot x$ or w^Tx
  - The **perceptron formula** becomes $\sigma(w \cdot x)$

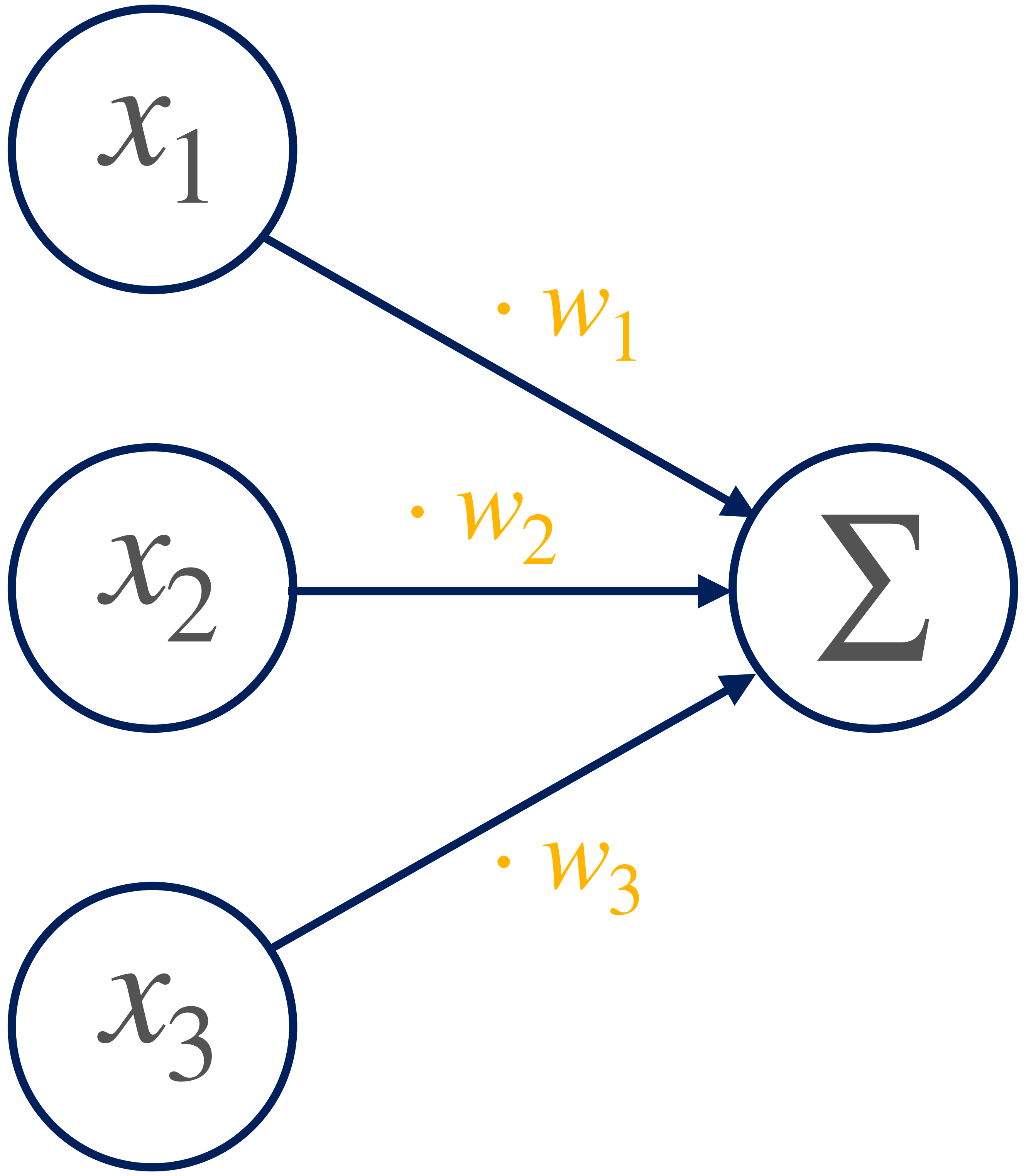


Vectors in the Perceptron

- The **weighted sum** of the perceptron...
  - takes each element of the **input vector** x...
  - times the element of the **weight vector** w:
  - $w_1x_1 + w_2x_2 + \ldots w_nx_n$
  - Where have we **seen this formula before?**
- This is the formula for the **dot product** of the two vectors!
  - $w \cdot x$ or w^Tx
  - The **perceptron formula** becomes $\sigma(w \cdot x)$
  - (where σ is the activation function)

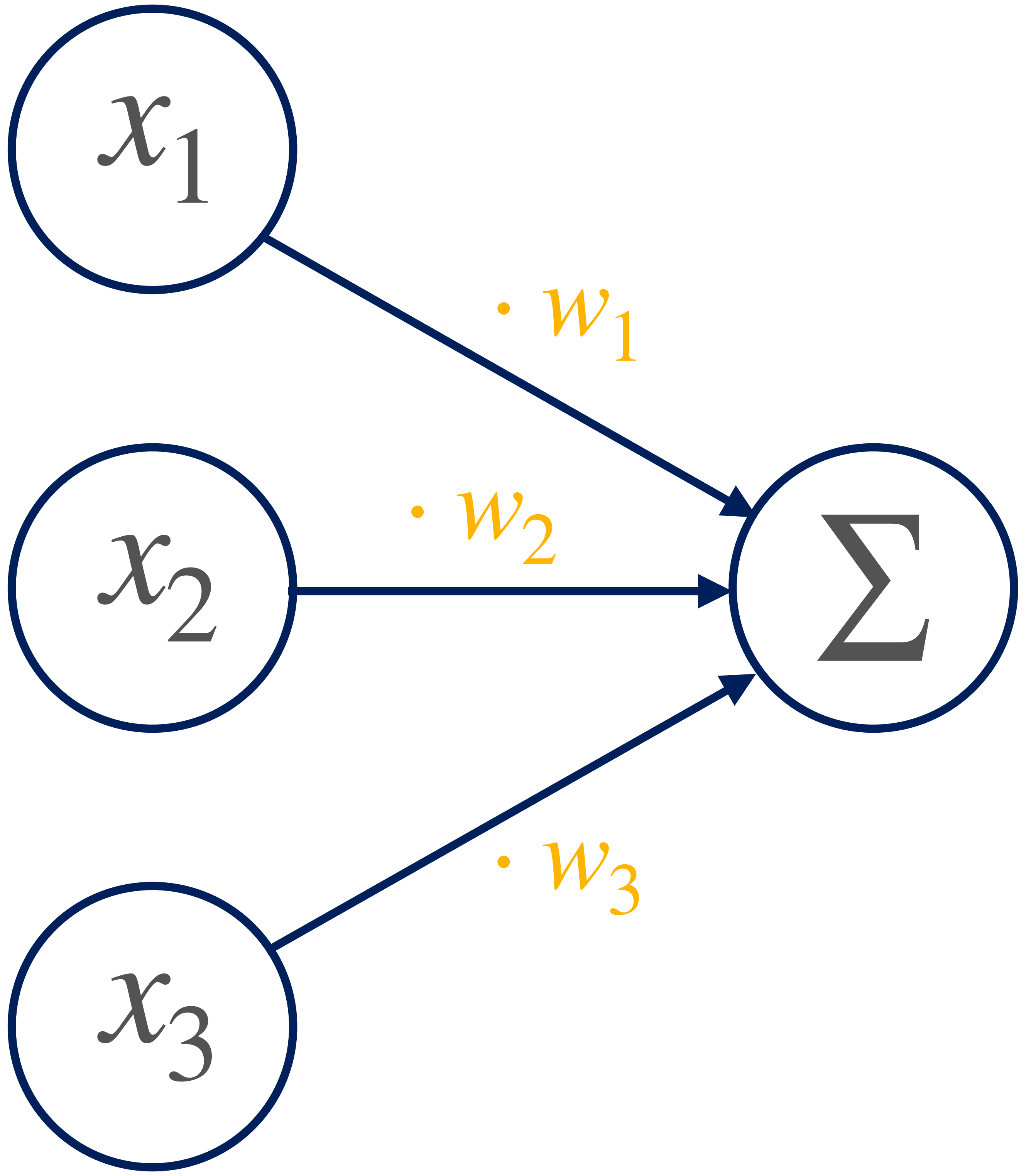


Perceptron Weights



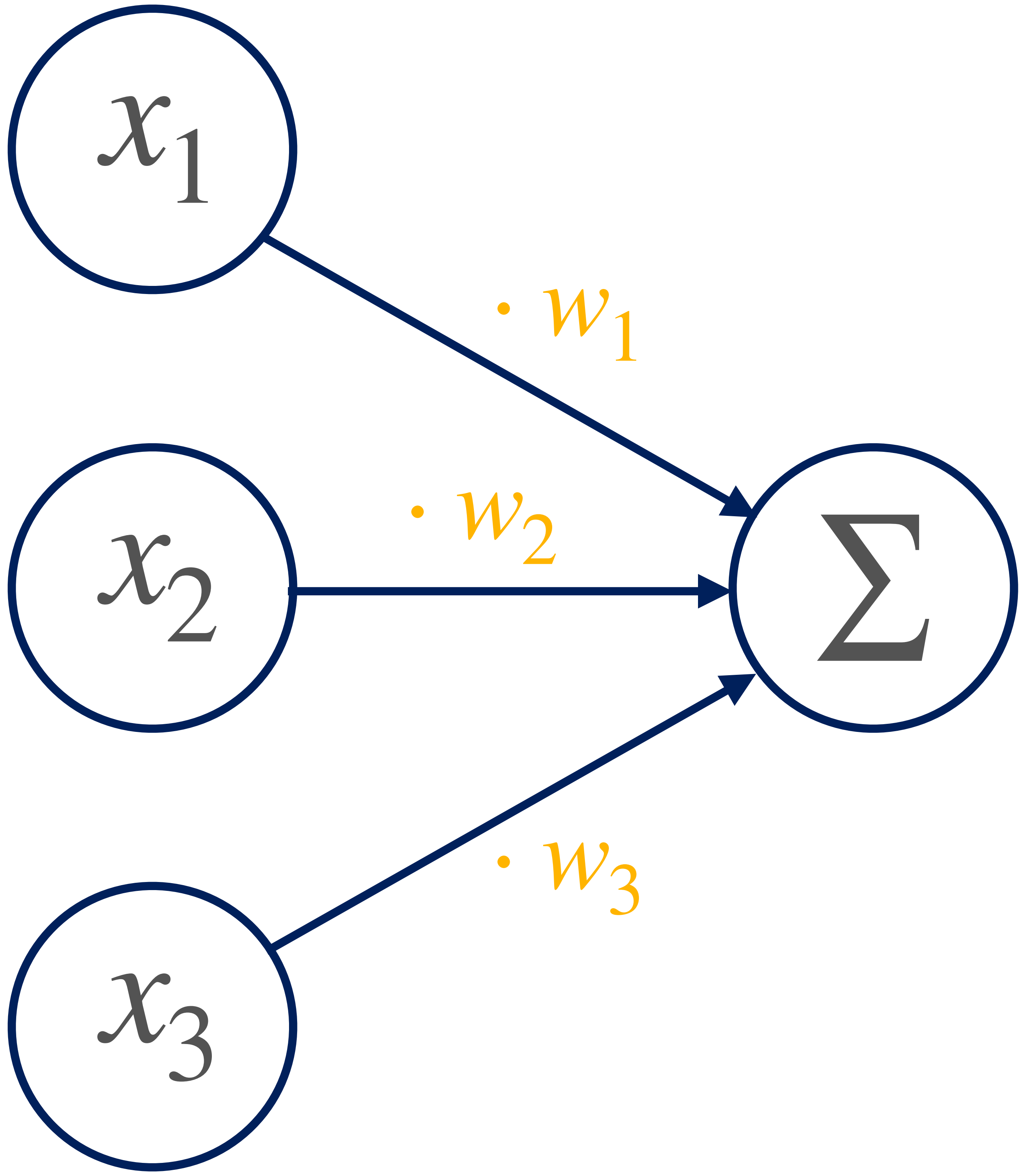
Perceptron Weights

- The input vector x is usually **defined by the data** that we want to model



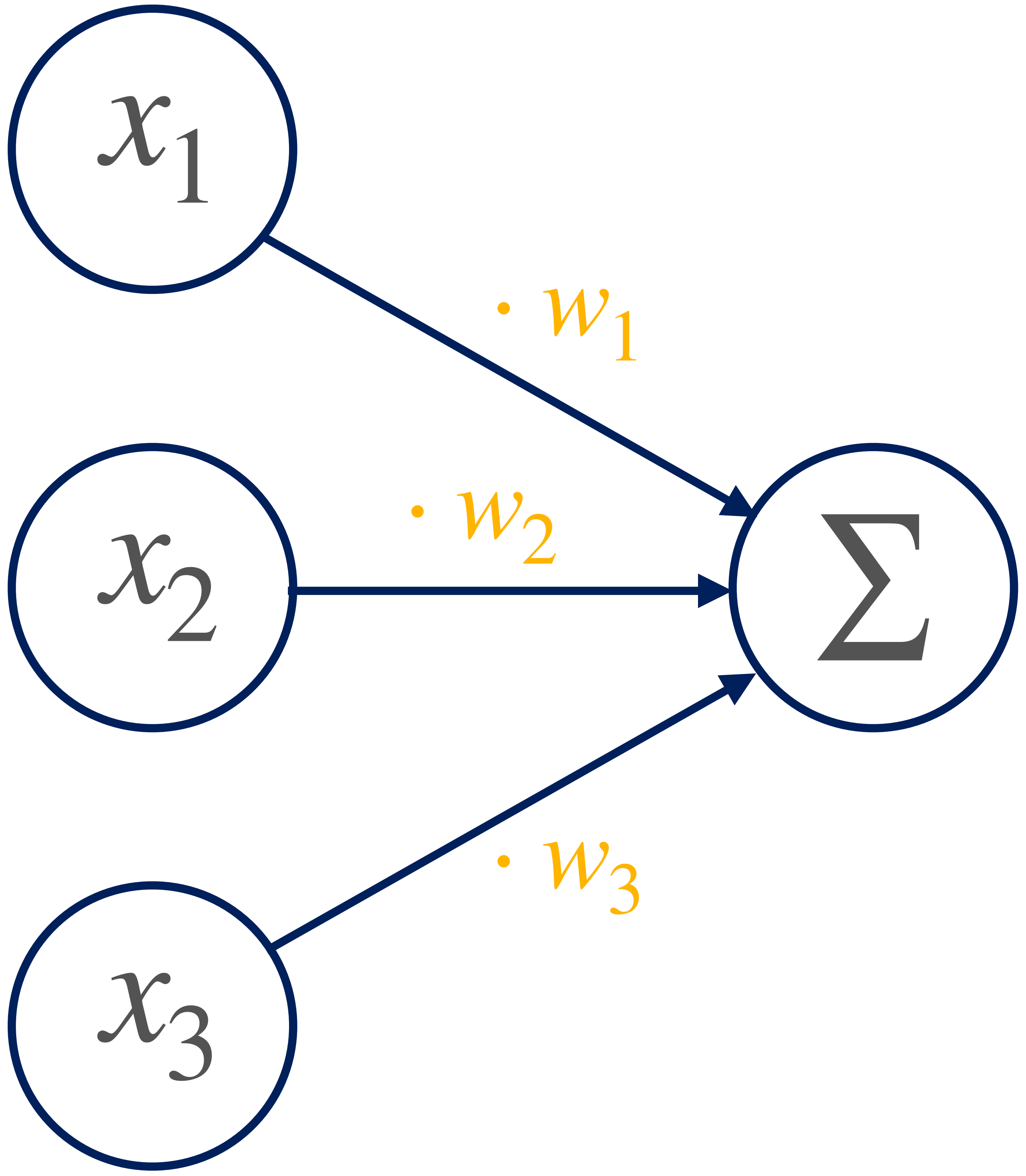
Perceptron Weights

- The input vector x is usually **defined by the data** that we want to model
- Where do the **weights w come from?**



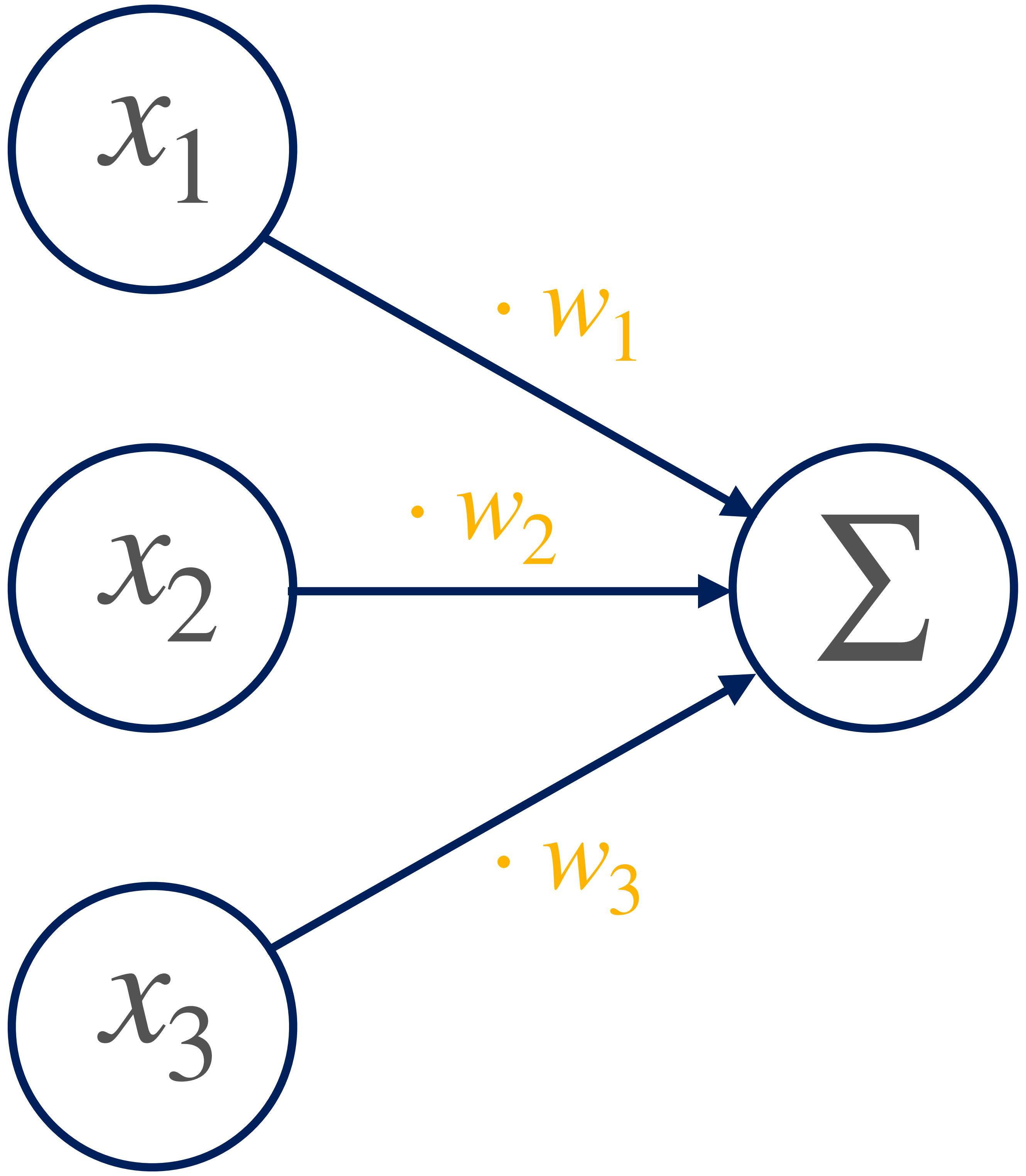
Perceptron Weights

- The input vector x is usually **defined by the data** that we want to model
- Where do the **weights w come from?**
  - These are usually **learned** by the model (i.e. "machine learning")

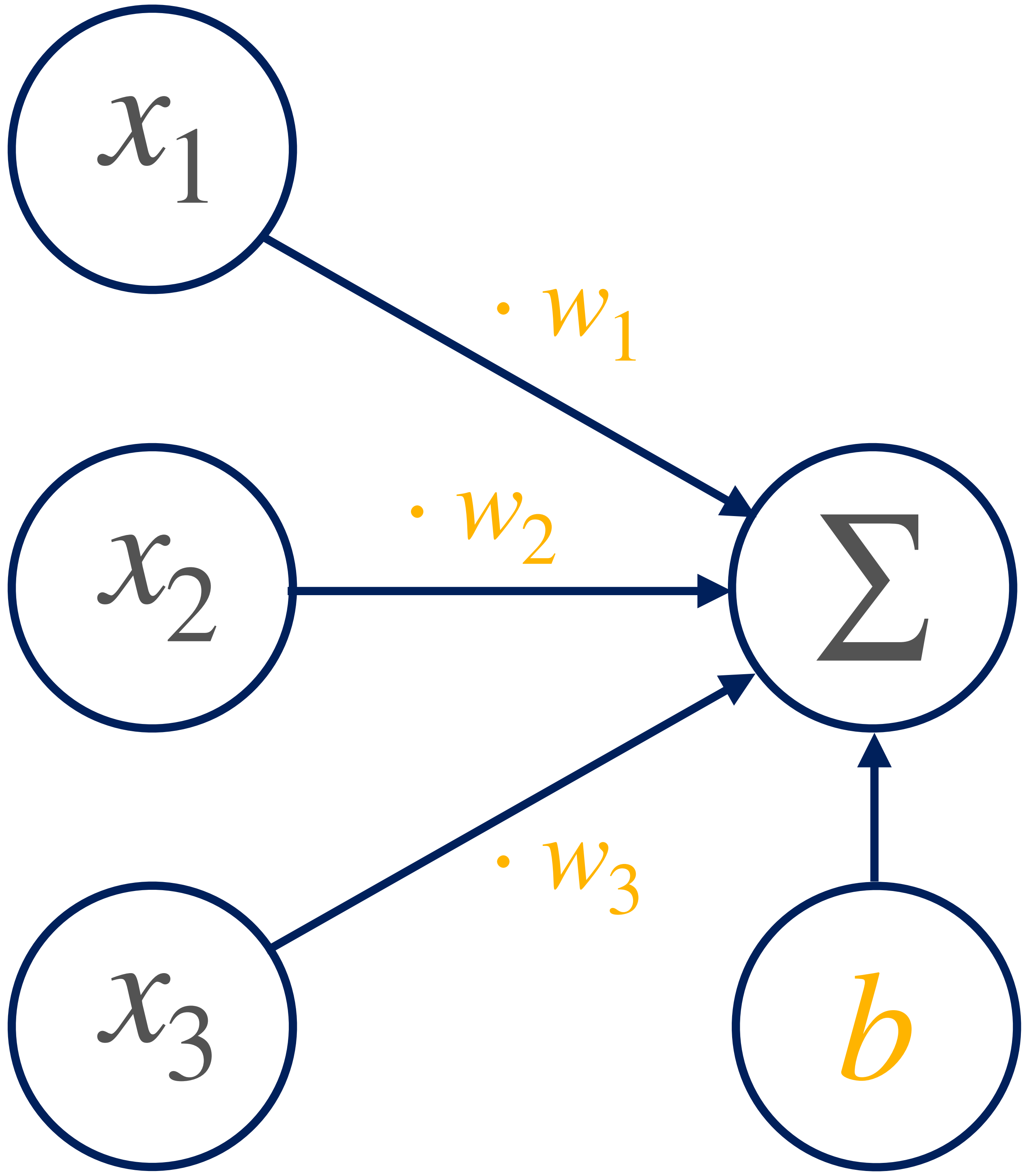


Perceptron Weights

- The input vector x is usually **defined by the data** that we want to model
- Where do the **weights w come from?**
  - These are usually **learned** by the model (i.e. "machine learning")
  - We'll introduce this learning next time

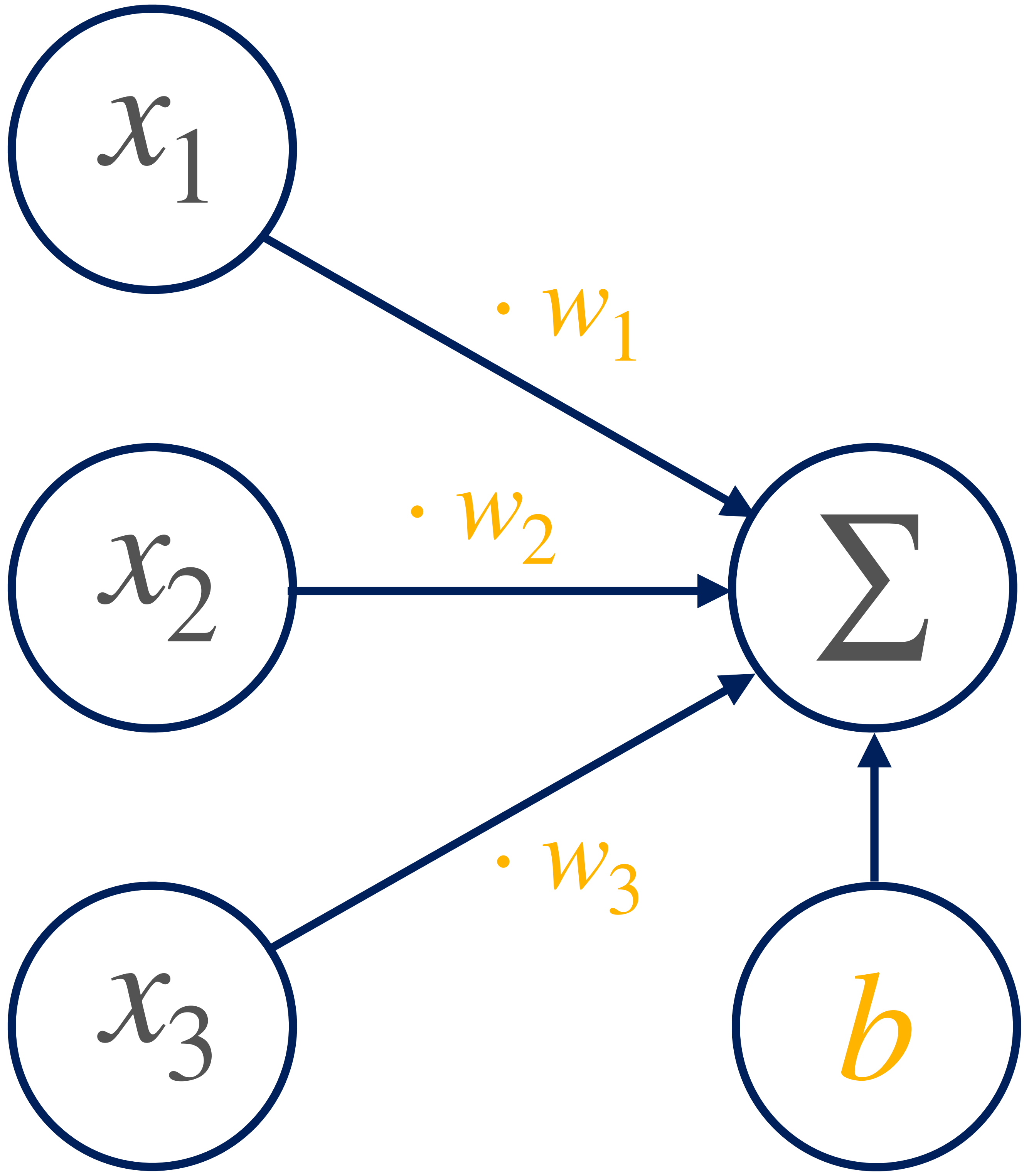


A Small Correction



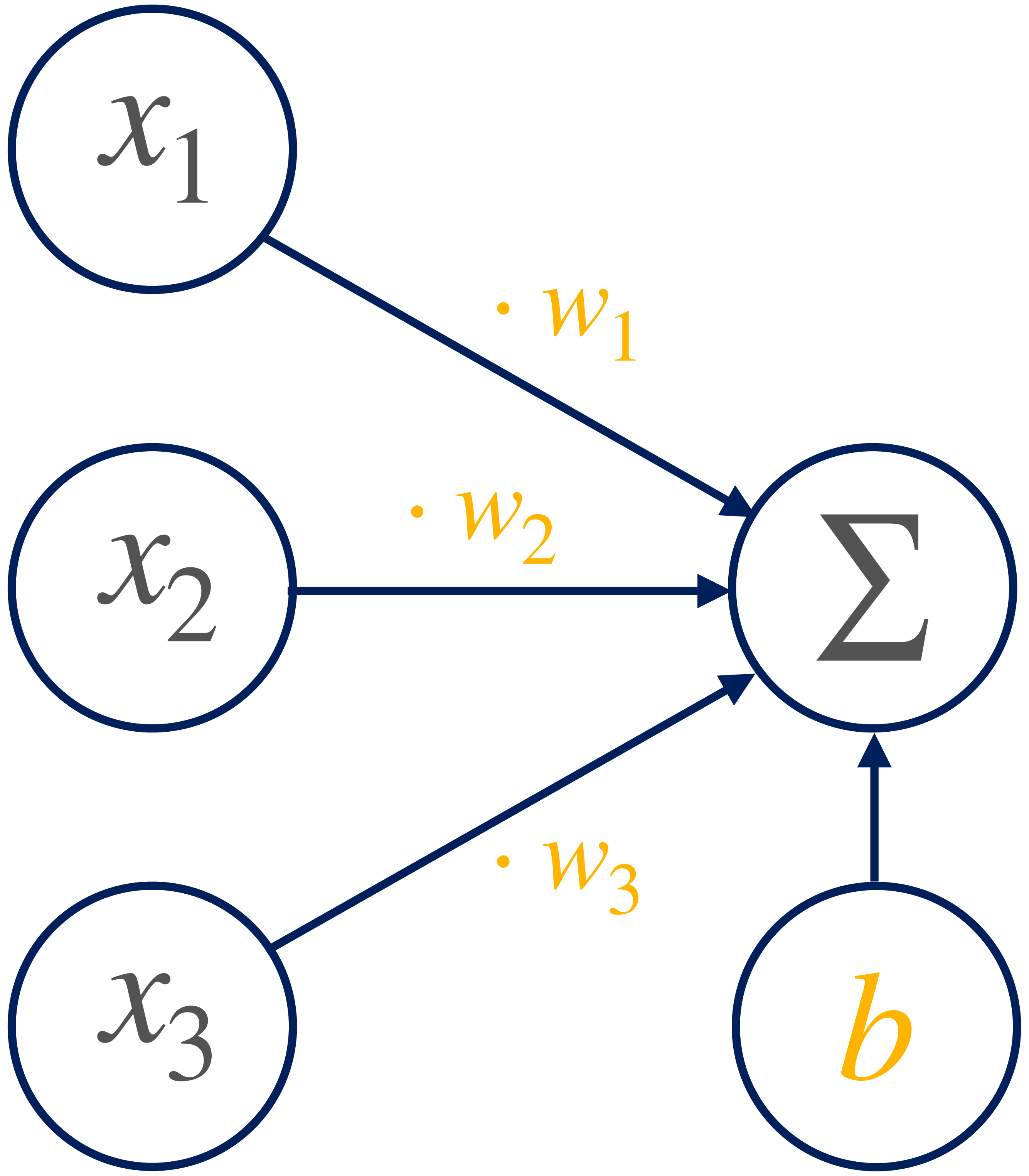
A Small Correction

- The perceptron has one more **learned value** called a **bias**



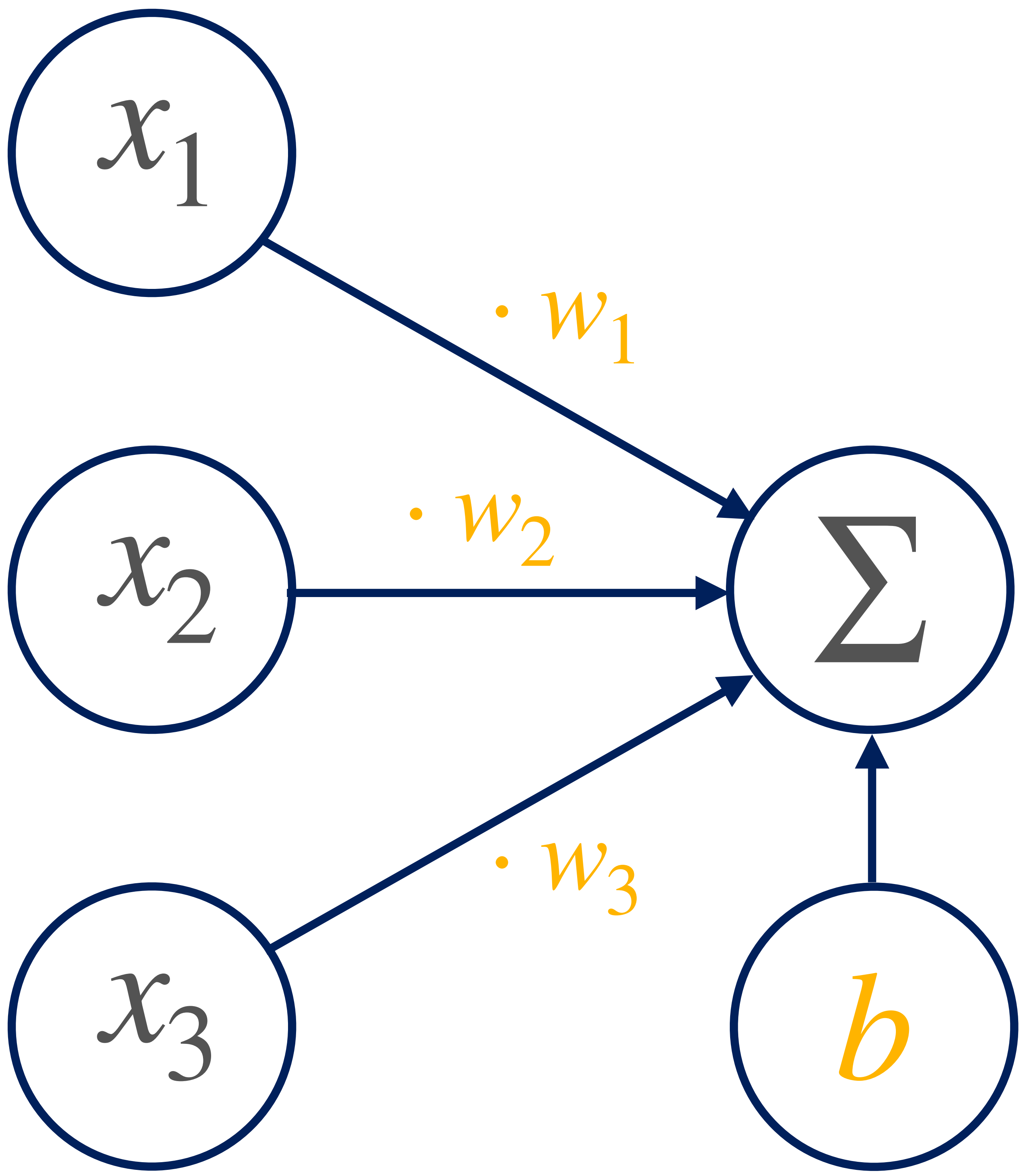
A Small Correction

- The perceptron has one more **learned value** called a **bias**
- The bias is **added to the summation**, and **does not correspond** to an input value

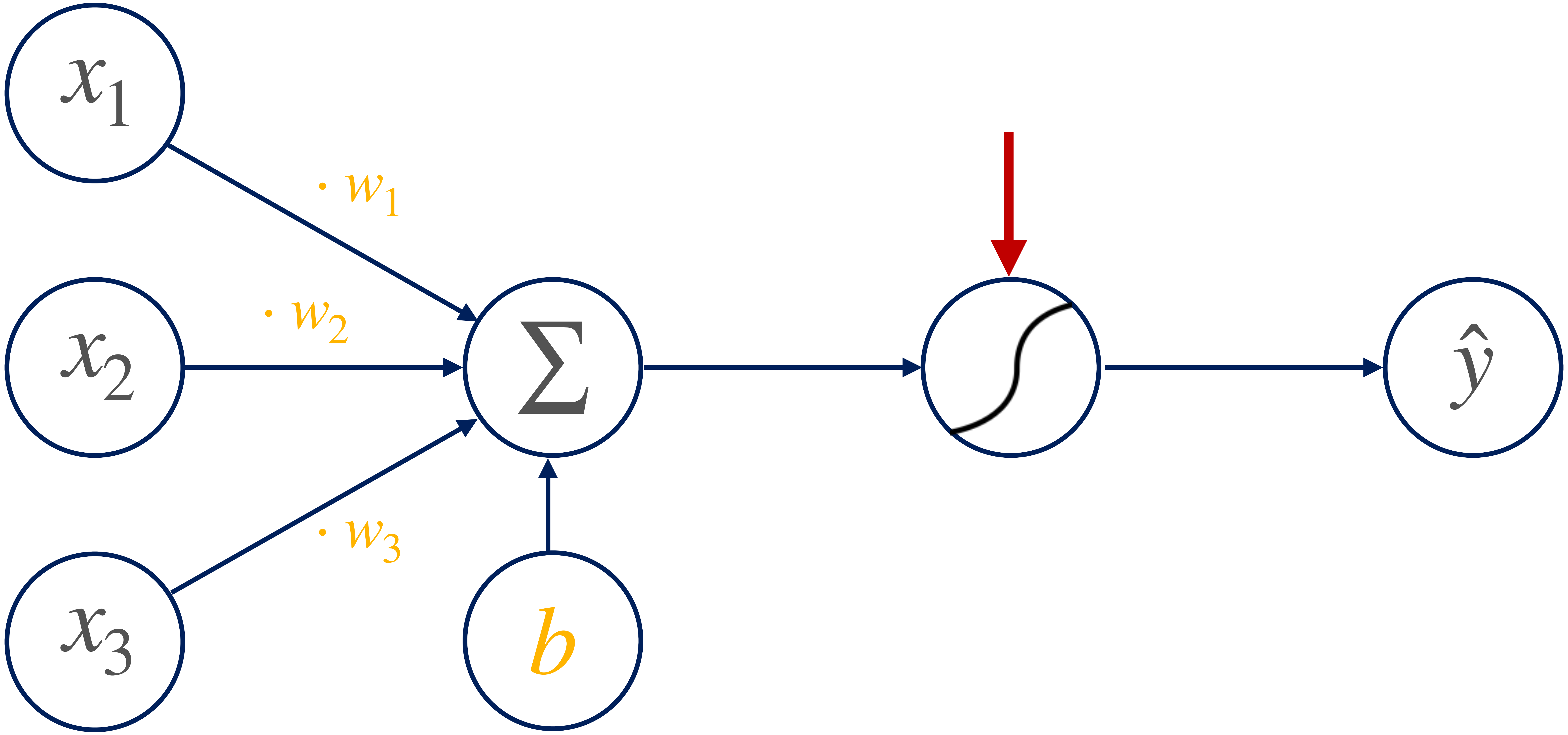


A Small Correction

- The perceptron has one more **learned value** called a **bias**
- The bias is **added to the summation**, and **does not correspond** to an input value
- Our updated formula is $\sigma(w \cdot x + b)$

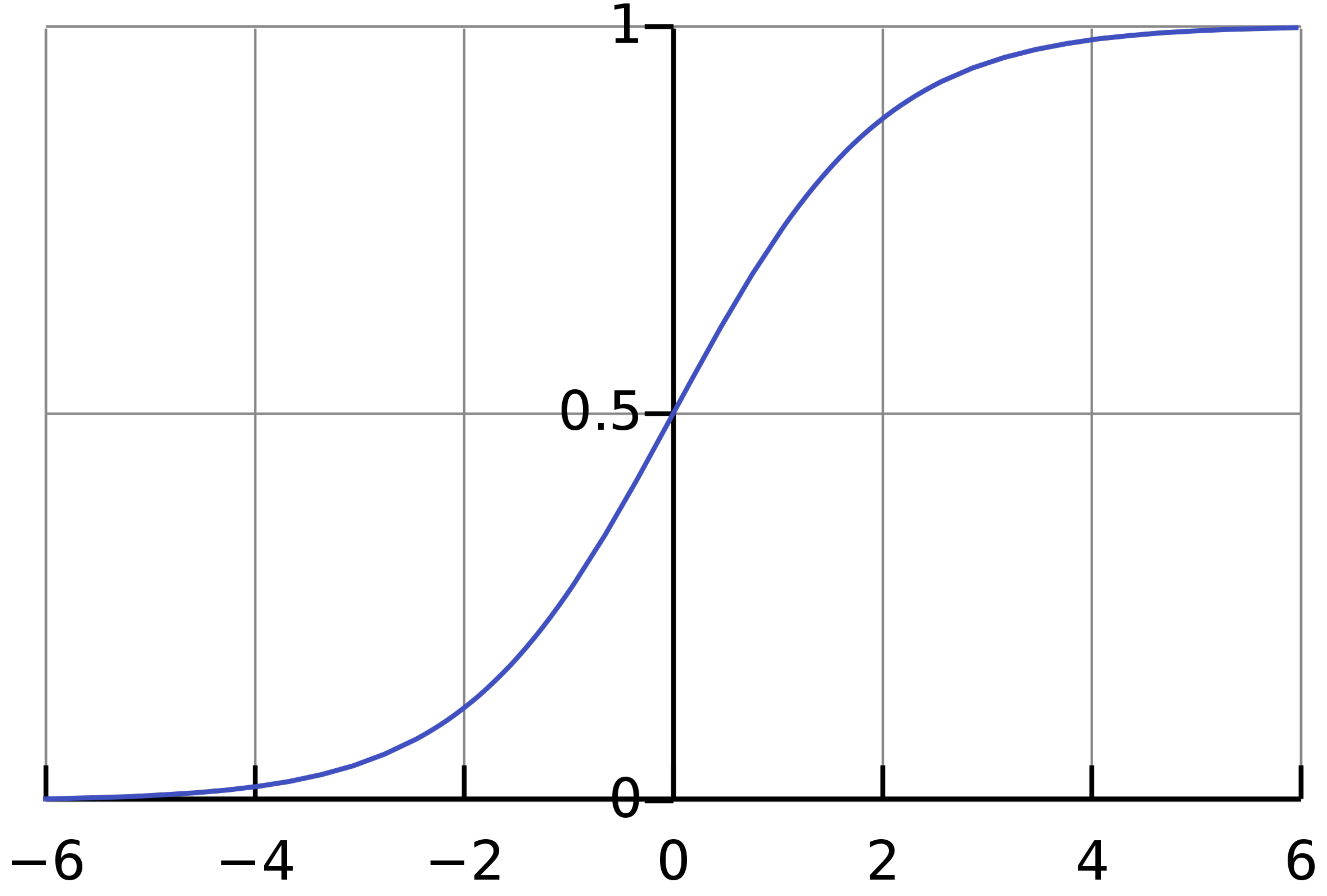


Introducing the Sigmoid



Sigmoid Function

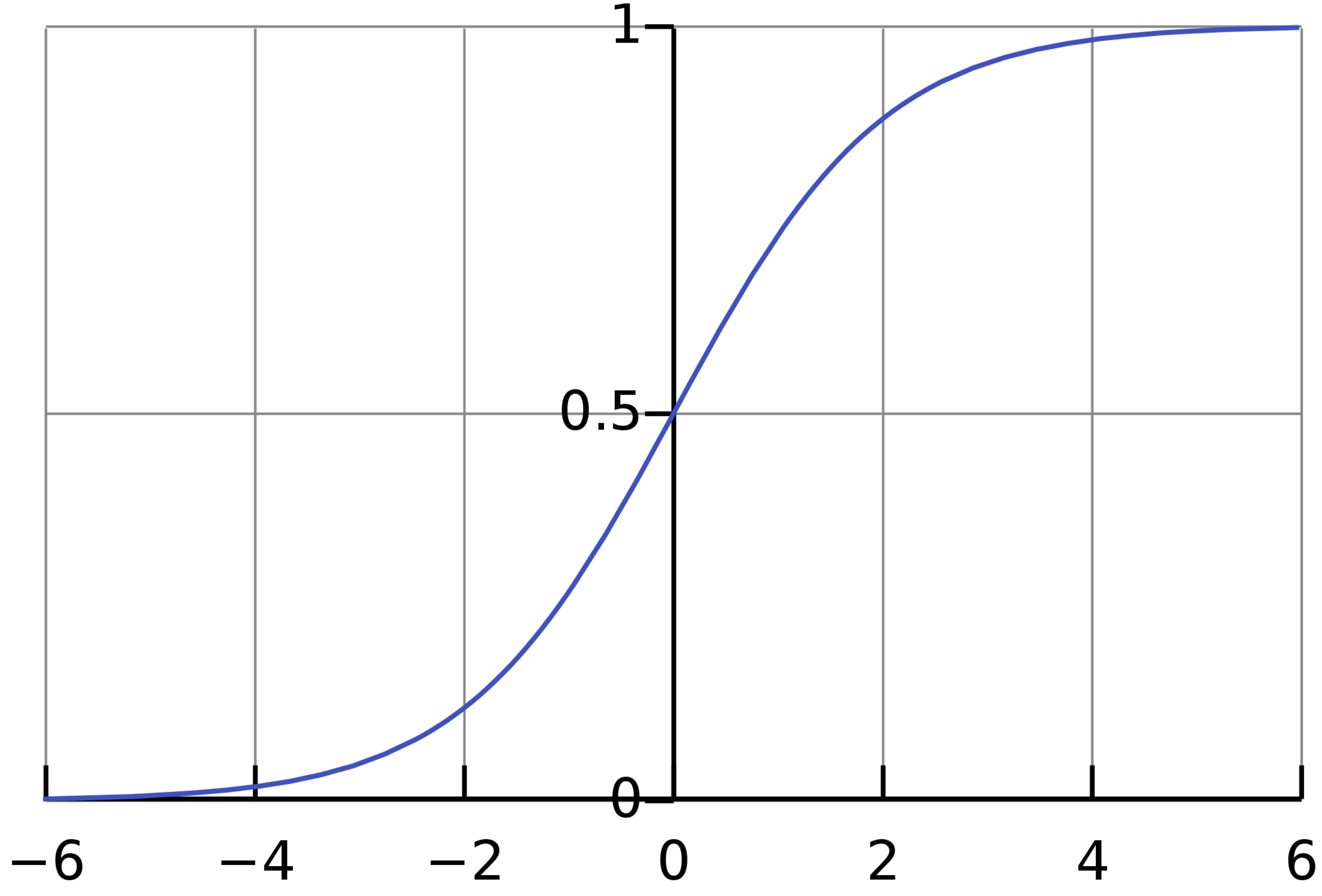
$$\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}$$



Sigmoid Function

- Called the "Sigmoid" due to its **S-like shape**
  - (Sigma σ is the Greek letter S)
  - A.K.A. the **Logistic Function**

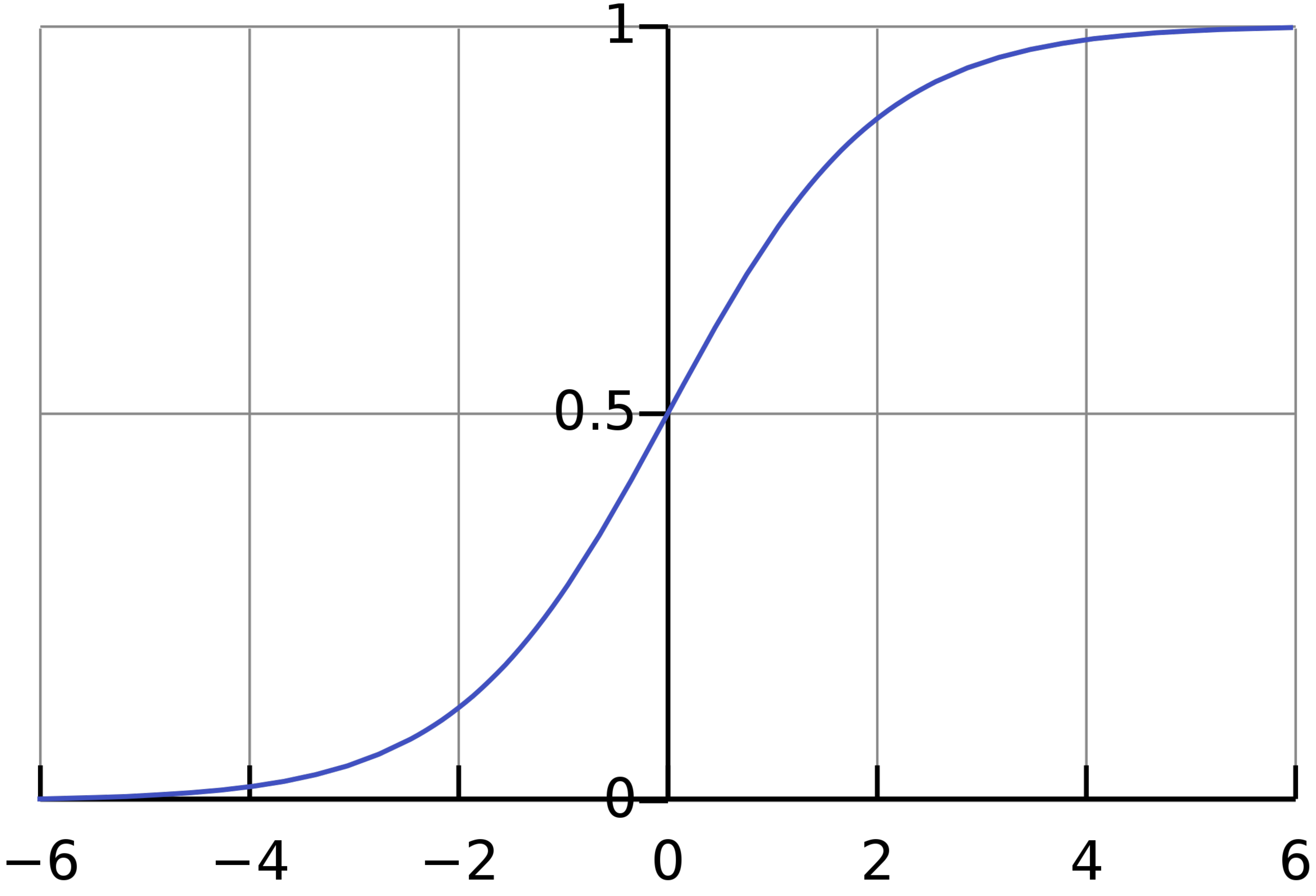
$$\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}$$



Sigmoid Function

- Called the "Sigmoid" due to its **S-like shape**
  - (Sigma σ is the Greek letter S)
  - A.K.A. the **Logistic Function**
- In Neural Networks, it fills the role of an **"activation function"** or **"non-linearity"** (more in a later lecture)

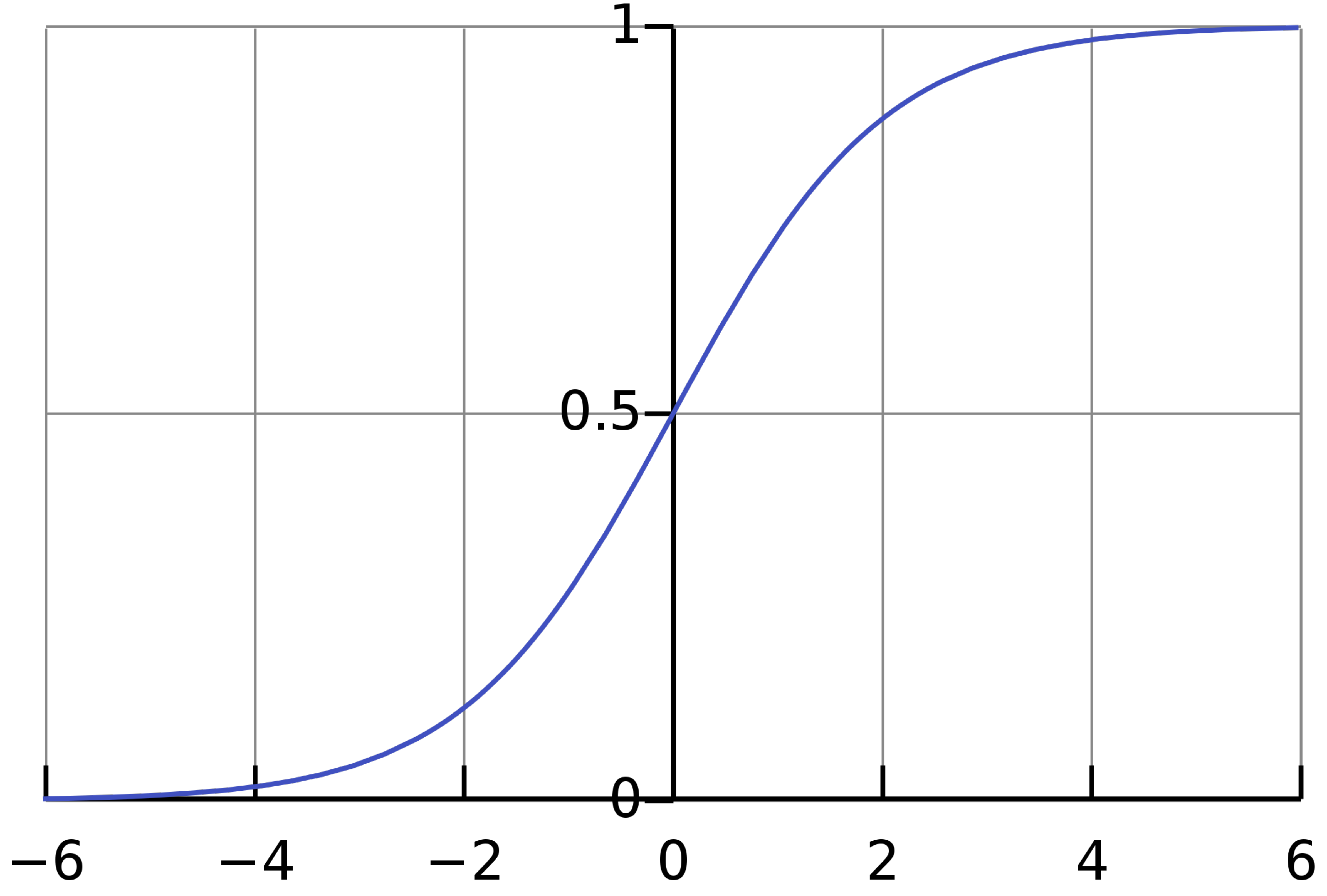
$$\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}$$



Sigmoid Function

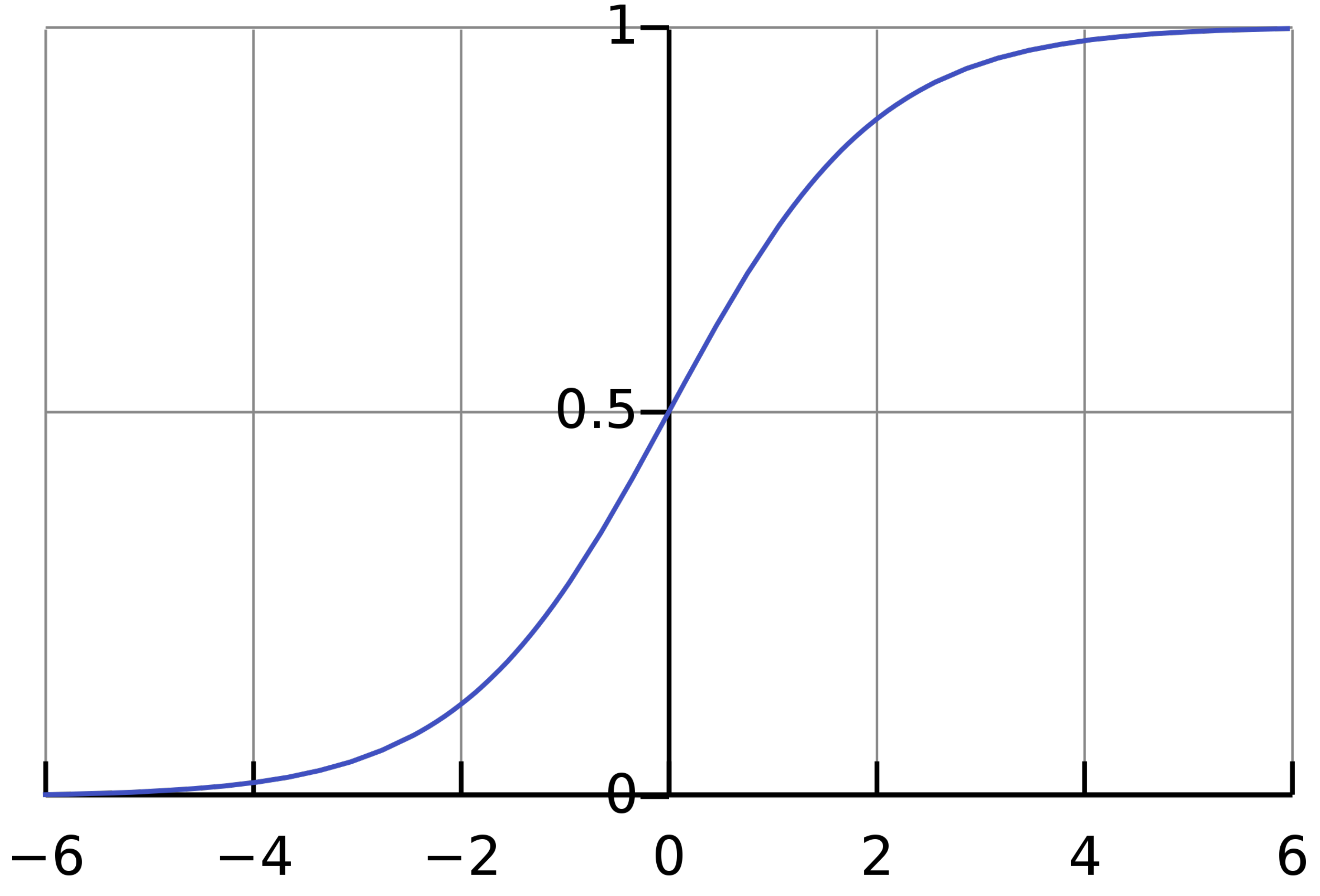
$$\sigma(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1}$$

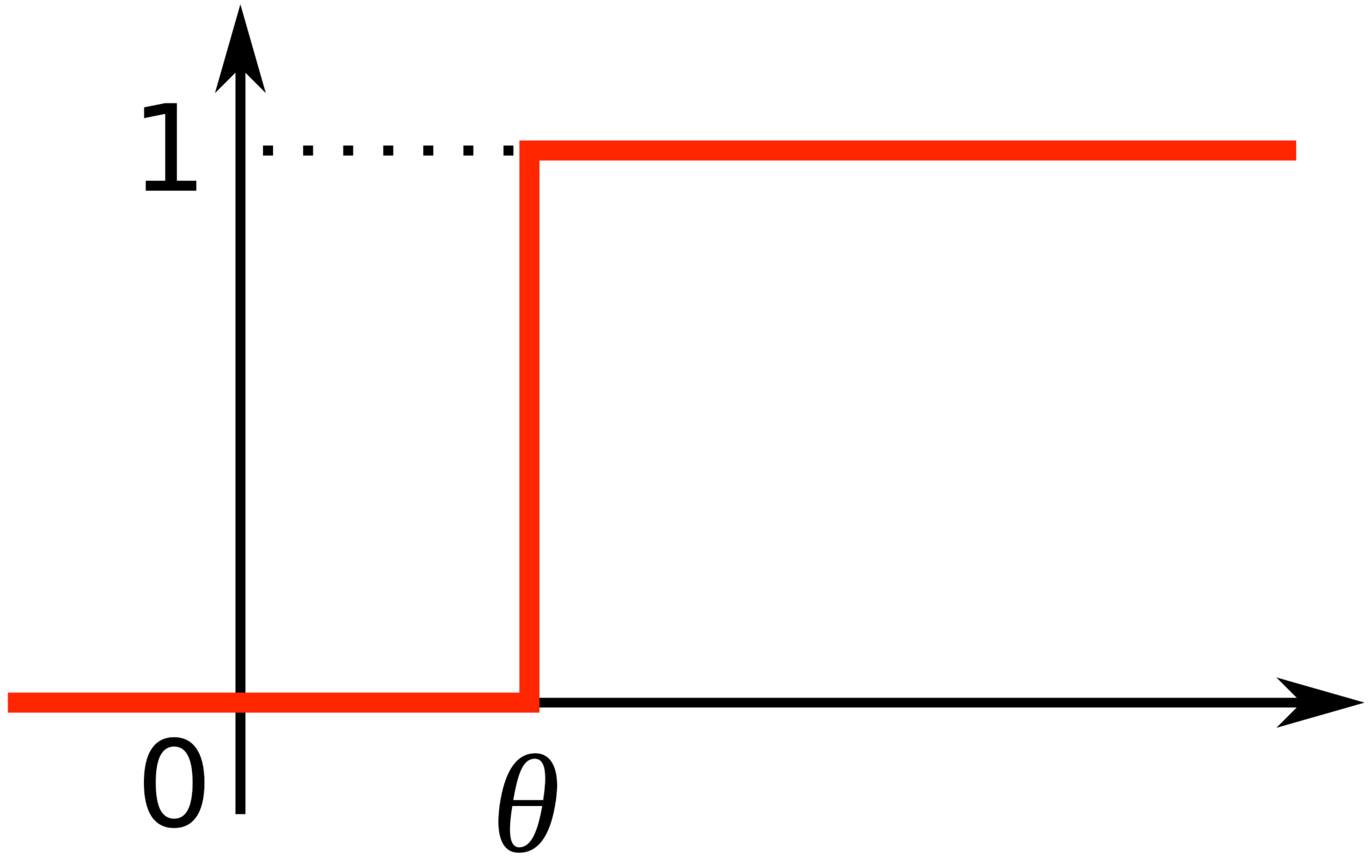
- Called the "Sigmoid" due to its **S-like shape**
  - (Sigma σ is the Greek letter S)
  - A.K.A. the **Logistic Function**
- In Neural Networks, it fills the role of an **"activation function"** or **"non-linearity"** (more in a later lecture)
- Works as a **soft threshold**
  - $\sigma(x)$ is **close to 1** when $x > 0$,
  - **close to 0** when $x < 0$
  - Output range: (0, 1)



Sigmoid vs. Hard Threshold

- Key difference: Sigmoid is **smooth**, i.e. has a **well-defined derivative**
- (This will be important when we get to **learning algorithms**)





Binary Classification

Binary Classification

- The sigmoid makes the Perceptron output $\hat{y}$ a **number between 0 and 1**
  - Usually viewed as a **probability** (i.e. 1.0 = 100%, 0.5 = 50%, etc.)

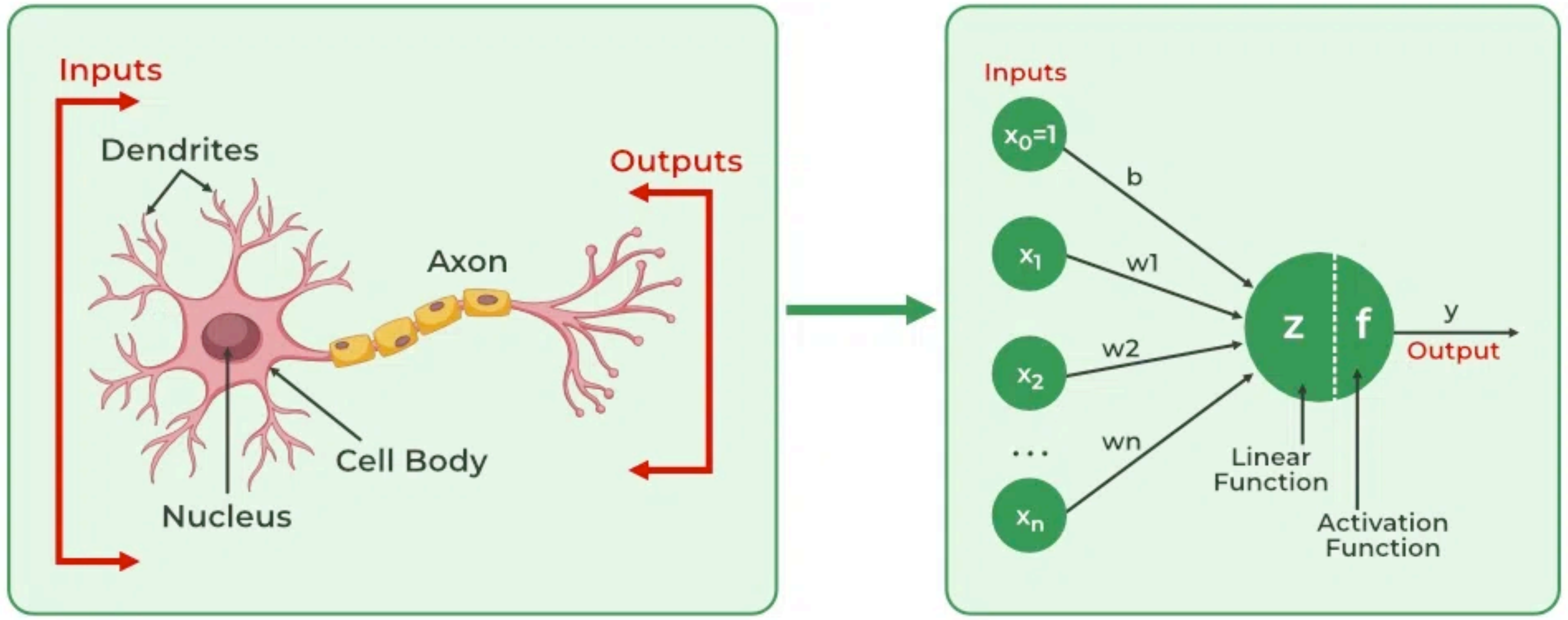
Binary Classification

- The sigmoid makes the Perceptron output $\hat{y}$ a **number between 0 and 1**
  - Usually viewed as a **probability** (i.e. 1.0 = 100%, 0.5 = 50%, etc.)
- Perceptron is used for **Binary Classification**
  - i.e. Positive vs. Negative, True vs. False, On vs. Off, 1 vs. 0
  - $\hat{y}$ is the **probability of positive classification** (according to the Perceptron)
  - $1 - \hat{y}$ is the **probability of negative classification**

Binary Classification

- The sigmoid makes the Perceptron output $\hat{y}$ a **number between 0 and 1**
  - Usually viewed as a **probability** (i.e. 1.0 = 100%, 0.5 = 50%, etc.)
- Perceptron is used for **Binary Classification**
  - i.e. Positive vs. Negative, True vs. False, On vs. Off, 1 vs. 0
  - $\hat{y}$ is the **probability of positive classification** (according to the Perceptron)
  - $1 - \hat{y}$ is the **probability of negative classification**
- Speeding ticket example: suppose $\mathit{Perceptron}(38) = \hat{y} = 0.92$
  - This means the Perceptron gives a **92% chance of getting a ticket**

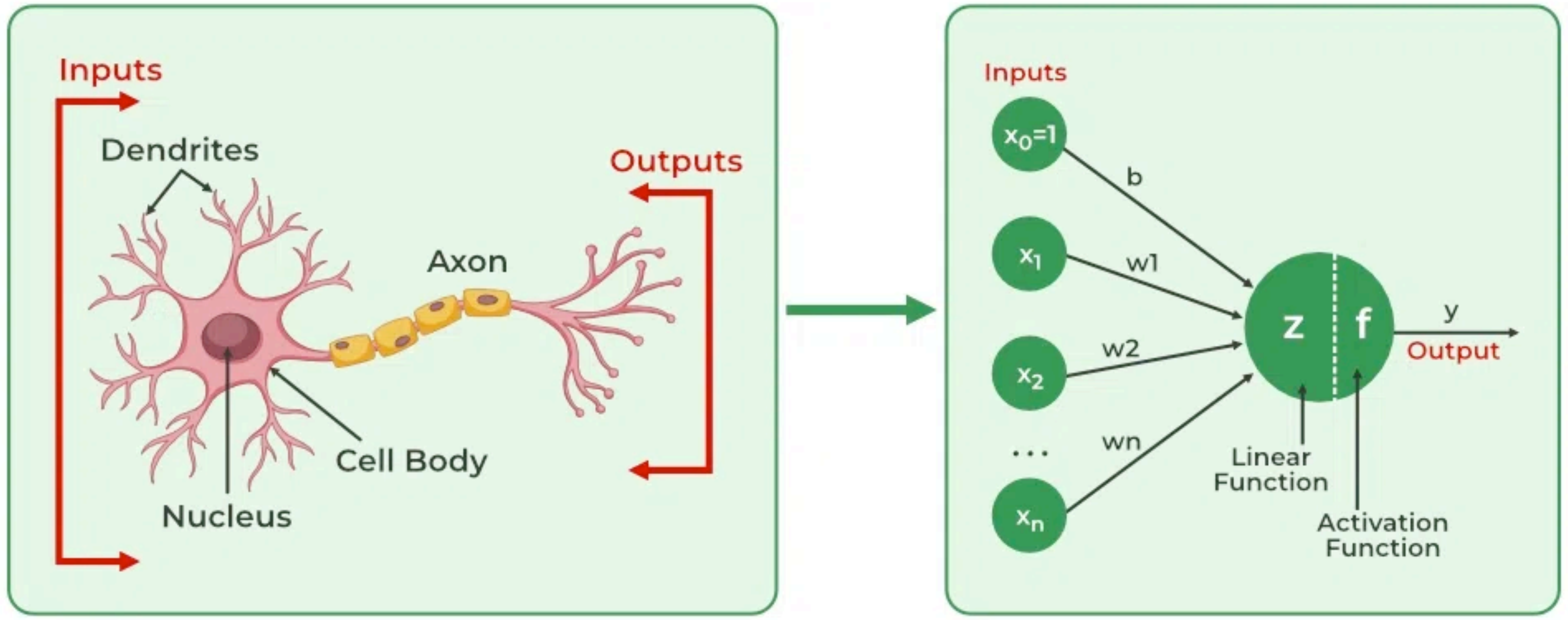
"Artificial Neuron"



source

"Artificial Neuron"

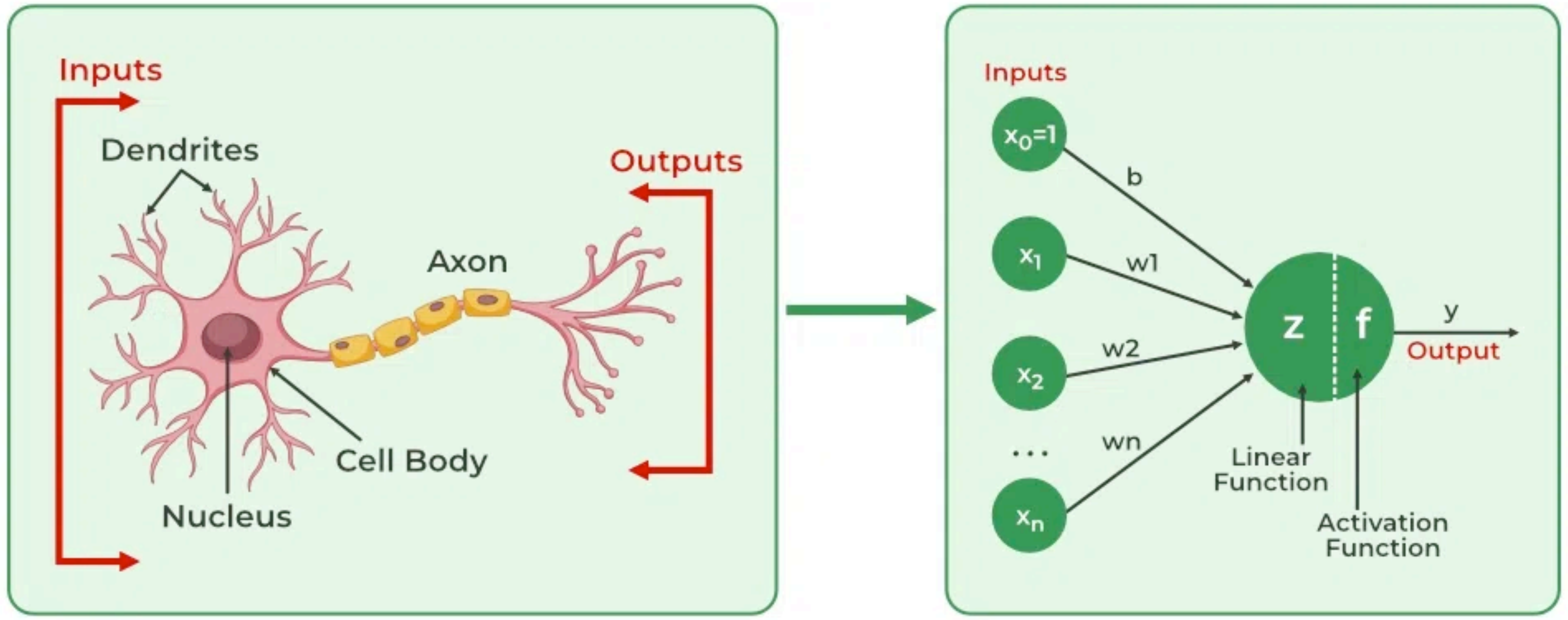
- The Perceptron is (loosely) inspired by a **human neuron** (brain cell)



source

"Artificial Neuron"

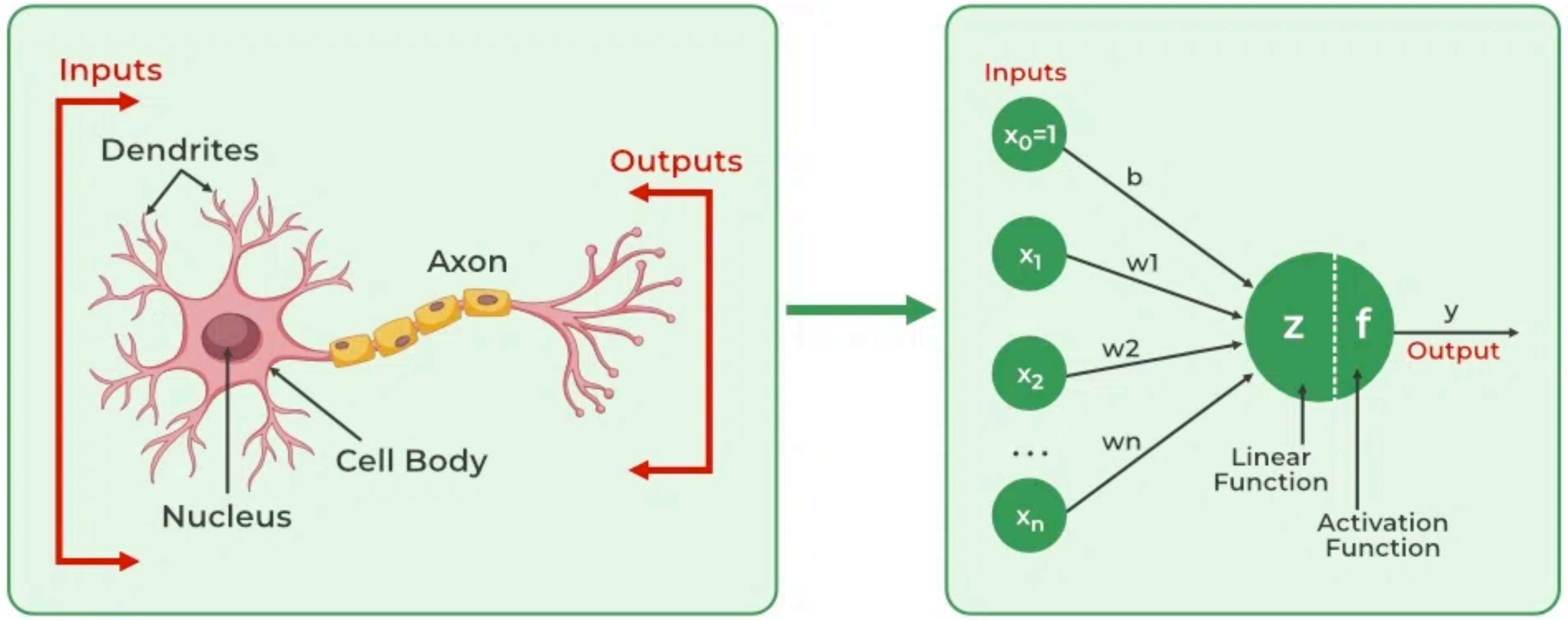
- The Perceptron is (loosely) inspired by a **human neuron** (brain cell)
- A neuron...
  - Takes in **input signals** from other neurons
  - **"Fires" an output signal** if the inputs **exceed some threshold**



source

"Artificial Neuron"

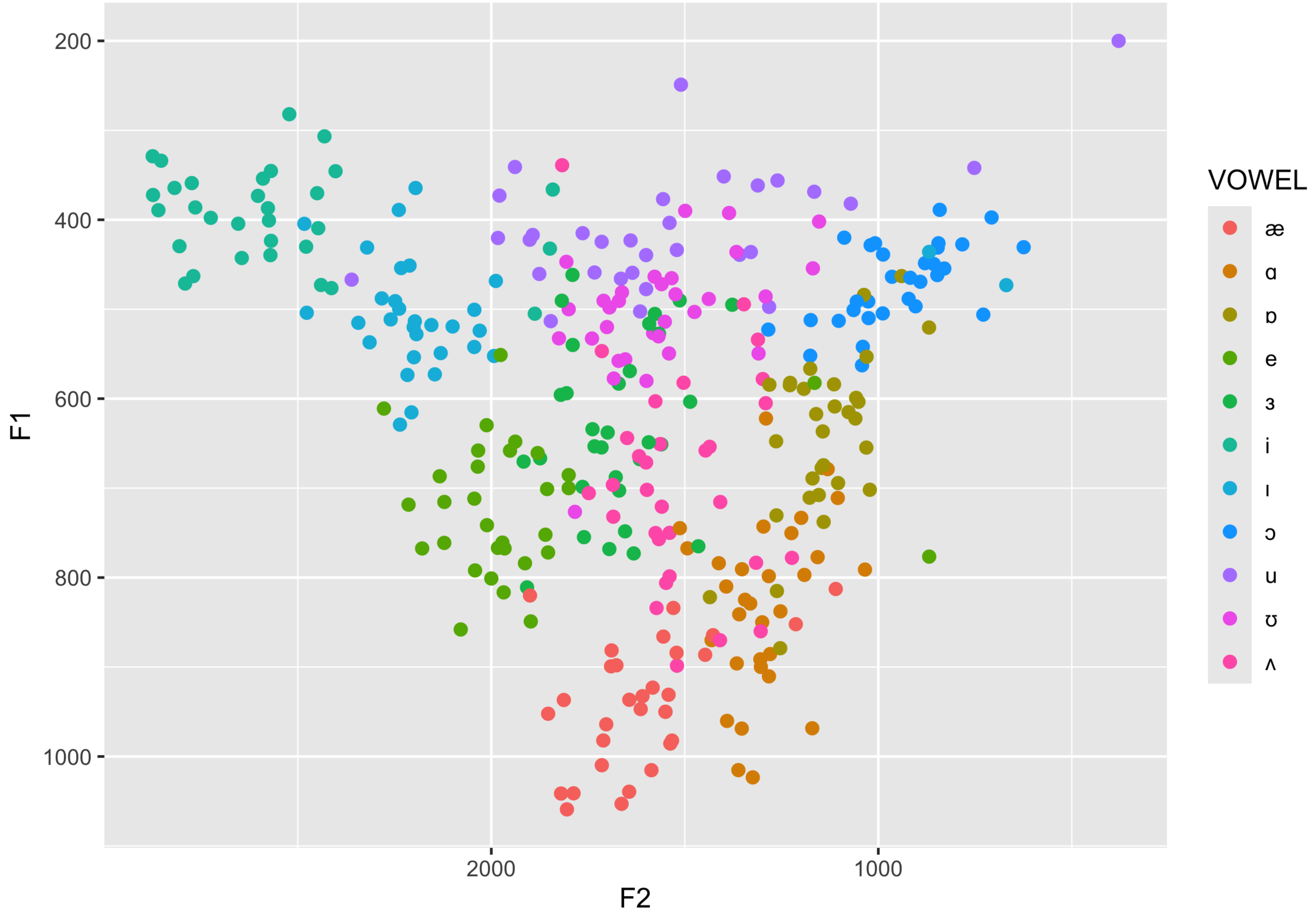
- The Perceptron is (loosely) inspired by a **human neuron** (brain cell)
- A neuron...
  - Takes in **input signals** from other neurons
  - **"Fires" an output signal** if the inputs **exceed some threshold**
- **Networks of neurons** can model sophisticated functions



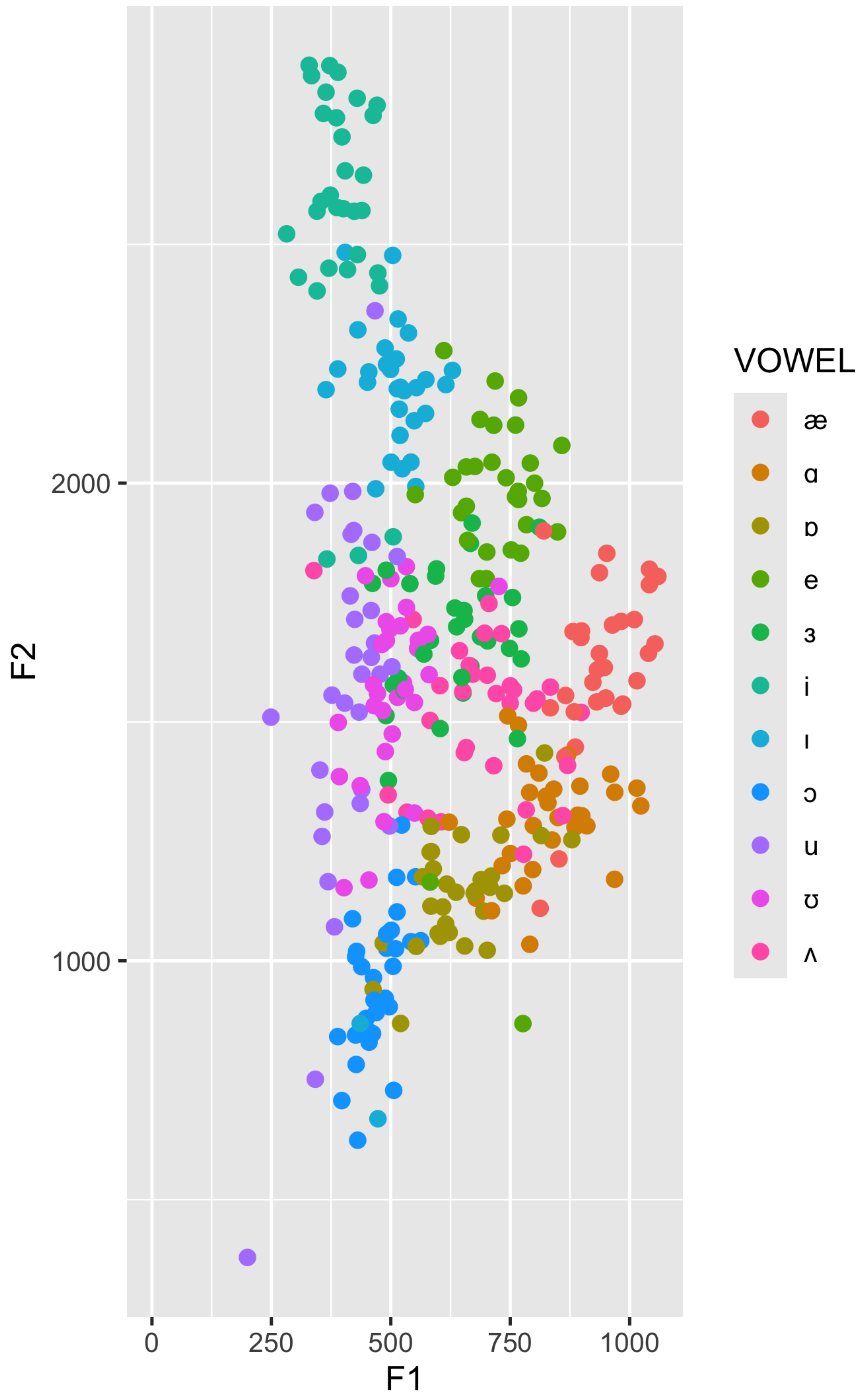
source

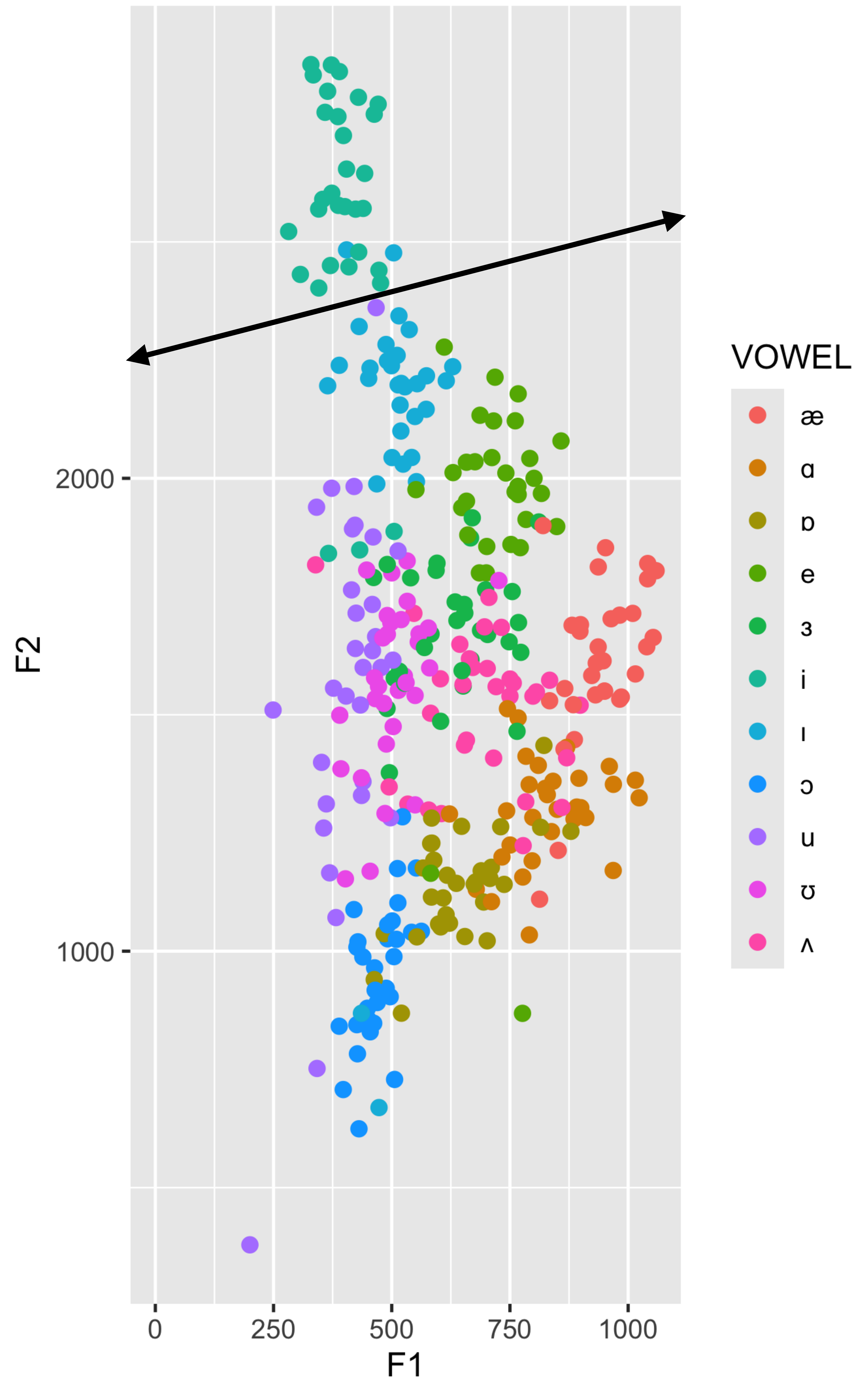
Perceptron and Linear Separation

- Example: **classifying a vowel** by its formant frequencies
- Vowels can be characterized by two component frequencies called **formants** (F1, F2)
  - These (very roughly) correspond to notions of "height" and "backness"
- Can we **identify the vowel [i]** with binary classification?



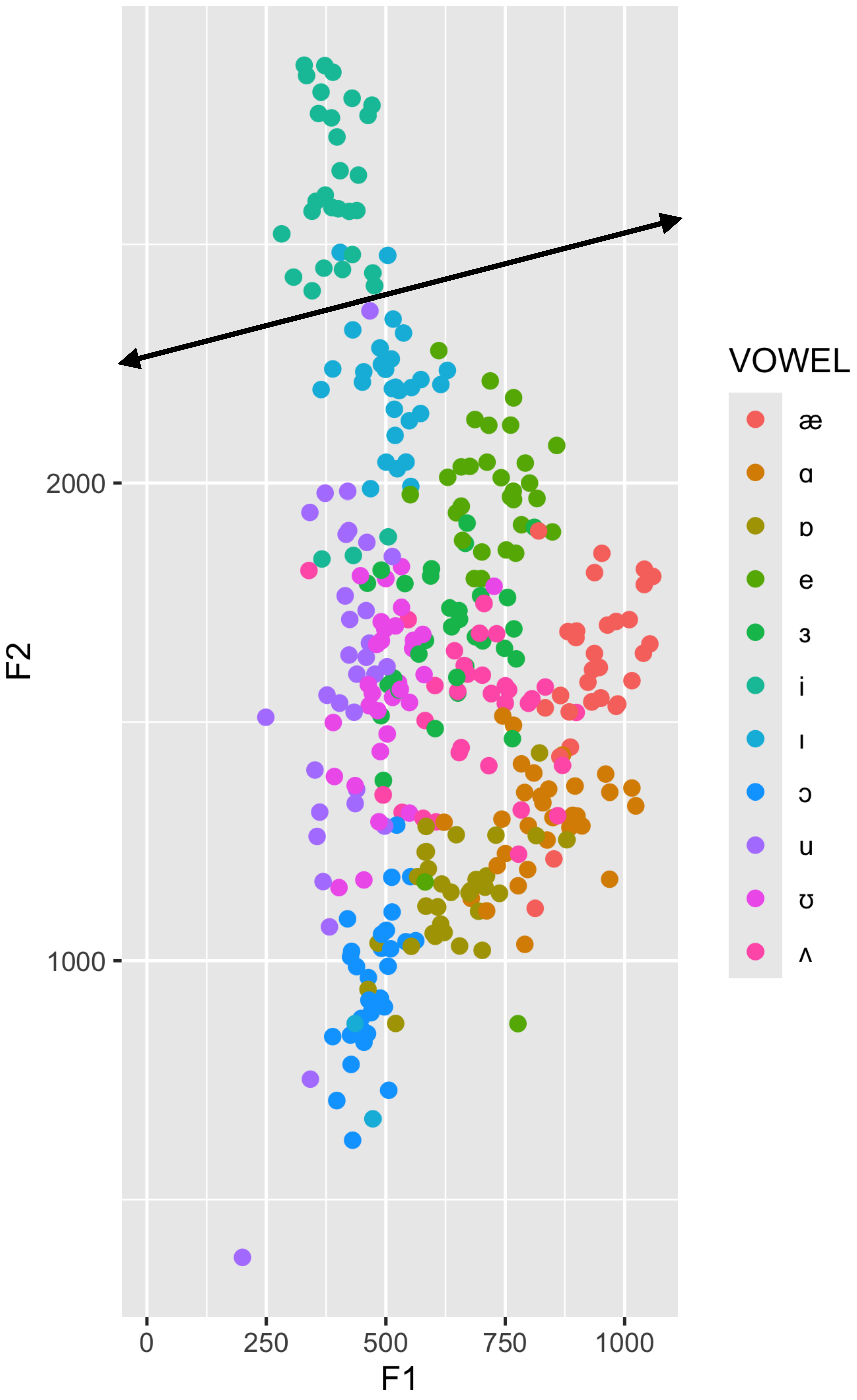
- Challenge: draw a **straight line** that **separates** the vowel [i] from **all other vowels**
  - Can a perfect boundary be drawn?
  - Draw an **imperfect one** if not
- Can we get a **formula** for this line?



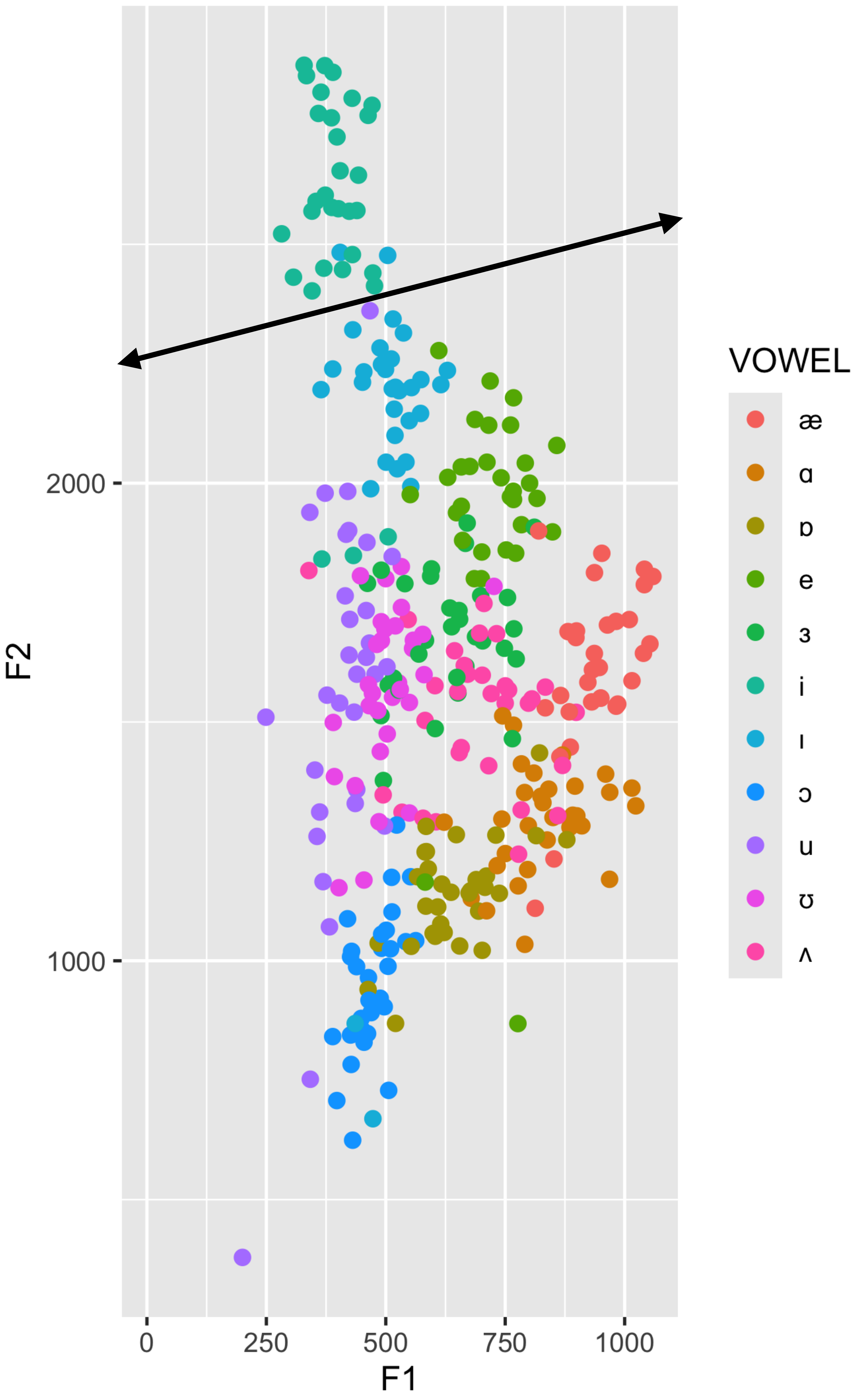


F2
2000
1000
0
250
500
750
1000
F1
VOWEL
æ
ɑ
ɒ
e
ɜ
i
ɪ
ɔ
u
ʊ
ʌ

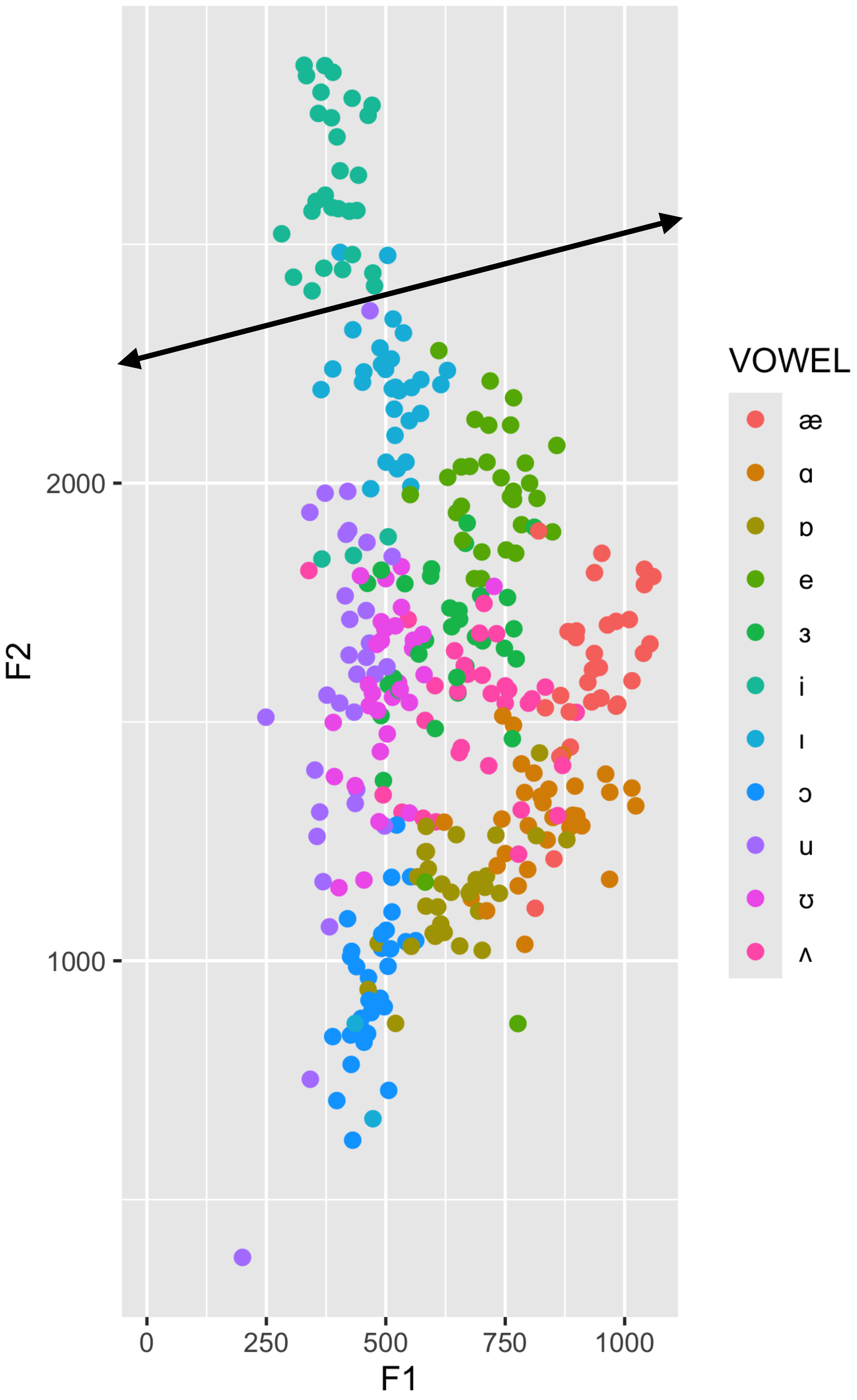
- We'd probably end up with something like this drawn line



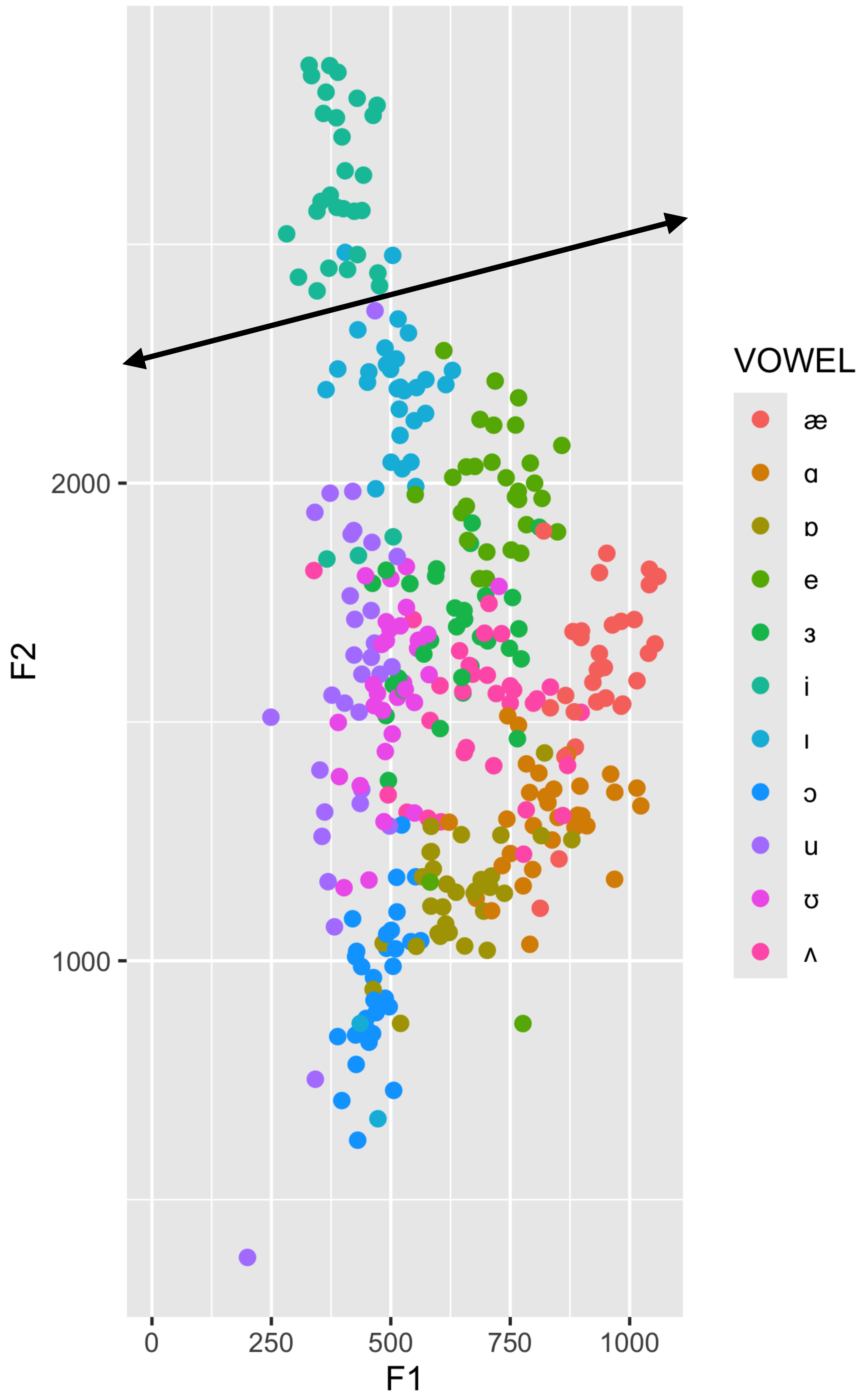
- We'd probably end up with something like this drawn line
- Line formula: $y = mx + b$



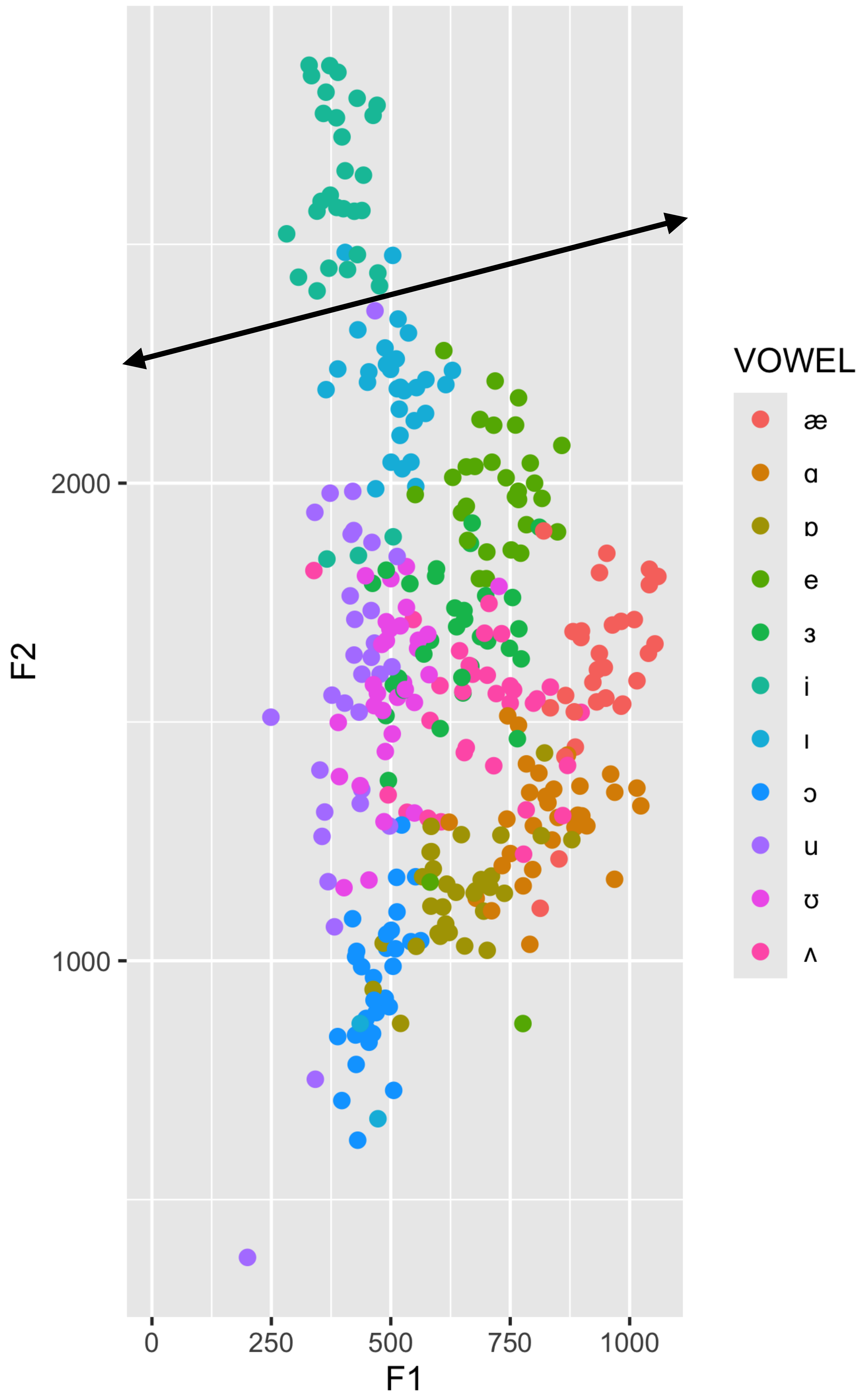
- We'd probably end up with something like this drawn line
- Line formula: $y = mx + b$
  - $\text{F2} = m \cdot \text{F1} + b$



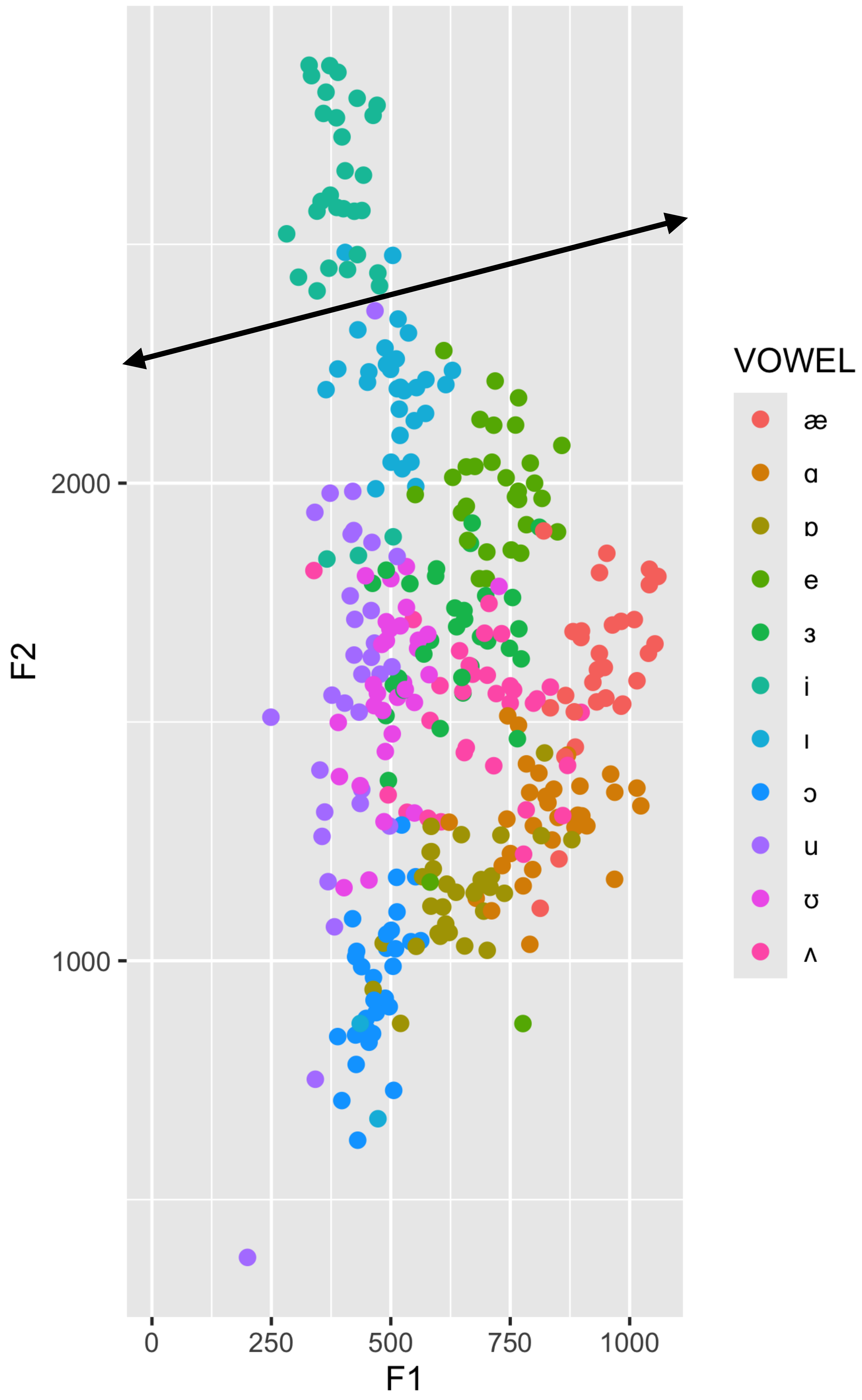
- We'd probably end up with something like this drawn line
- Line formula: $y = mx + b$
  - $\text{F2} = m \cdot \text{F1} + b$
  - $b \approx 2265$ (y-intercept)



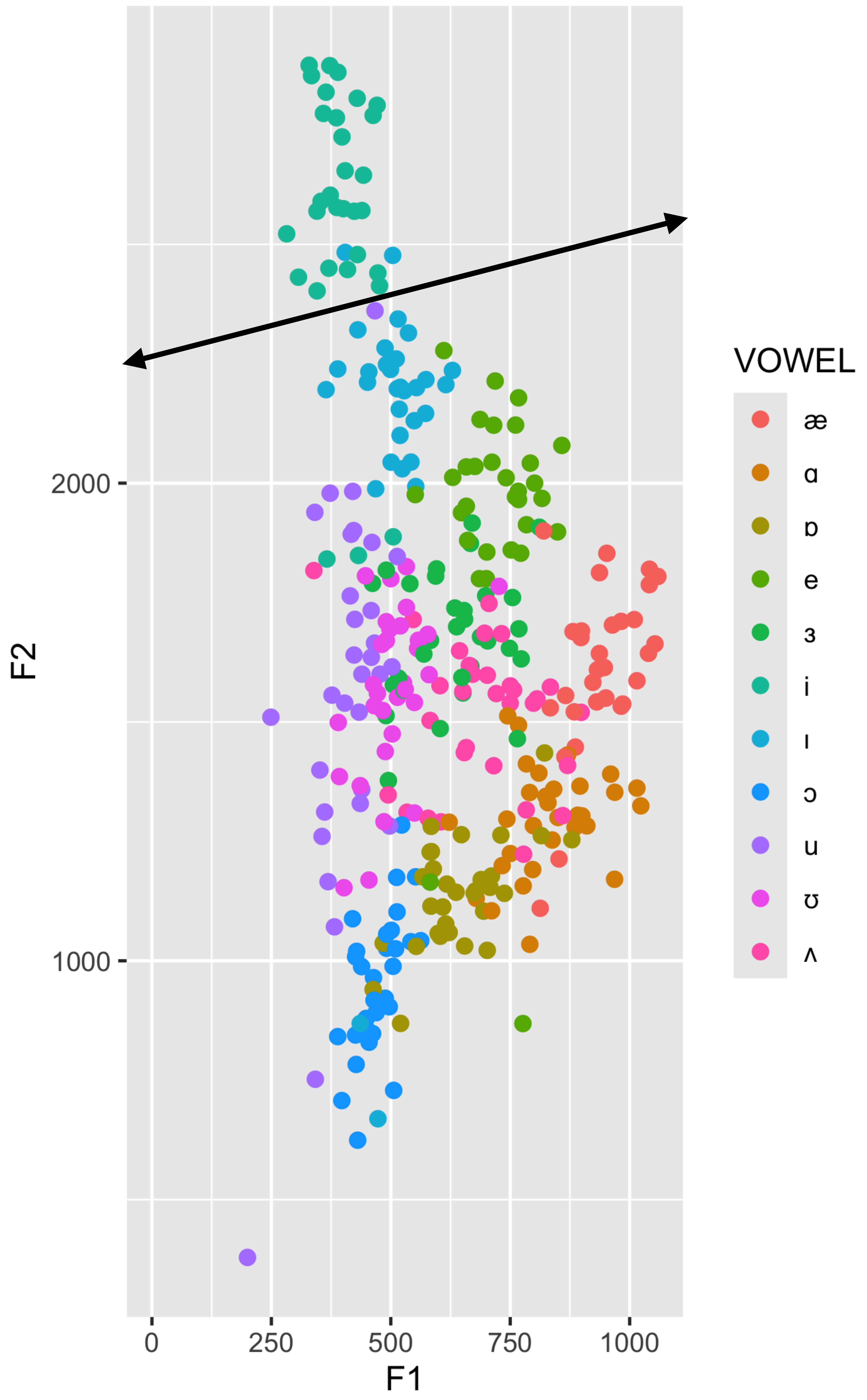
- We'd probably end up with something like this drawn line
- Line formula: $y = mx + b$
  - F2 $= m \cdot$ F1 $+ b$
  - $b \approx 2265$ (y-intercept)
  - $m \approx 294/1125$ (slope, "rise over run")



- We'd probably end up with something like this drawn line
- Line formula: $y = mx + b$
  - $\text{F2} = m \cdot \text{F1} + b$
  - $b \approx 2265$ (y-intercept)
  - $m \approx 294/1125$ (slope, "rise over run")
  - $\text{F2} = \dfrac{294}{1125}\text{F1} + 2265$



- We'd probably end up with something like this drawn line
- Line formula: $y = mx + b$
  - $\text{F2} = m \cdot \text{F1} + b$
  - $b \approx 2265$ (y-intercept)
  - $m \approx 294/1125$ (slope, "rise over run")
  - $\text{F2} = \dfrac{294}{1125}\text{F1} + 2265$
- Let's rearrange this:

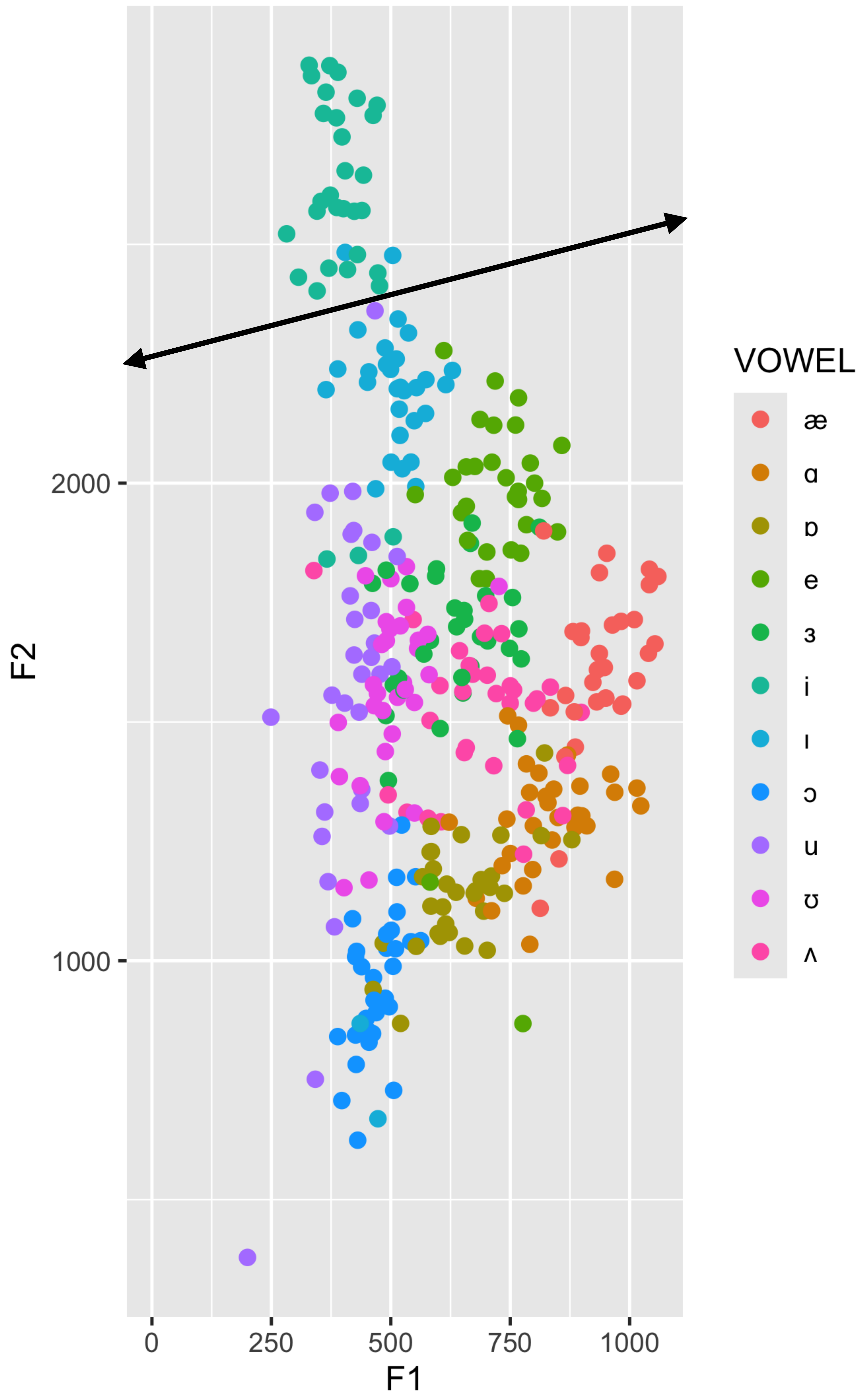


- We'd probably end up with something like this drawn line
- Line formula: $y = mx + b$
  - $\text{F2} = m \cdot \text{F1} + b$
  - $b \approx 2265$ (y-intercept)
  - $m \approx 294/1125$ (slope, "rise over run")
  - $\text{F2} = \dfrac{294}{1125}\text{F1} + 2265$
- Let's rearrange this:

$$\text{F2} - \frac{294}{1125}\text{F1} - 2265 = 0$$

$$1.0 \cdot \text{F2} + (-0.26) \cdot \text{F1} - 2265 = 0$$

$$-0.26 \cdot \text{F1} + 1.0 \cdot \text{F2} - 2265 = 0$$



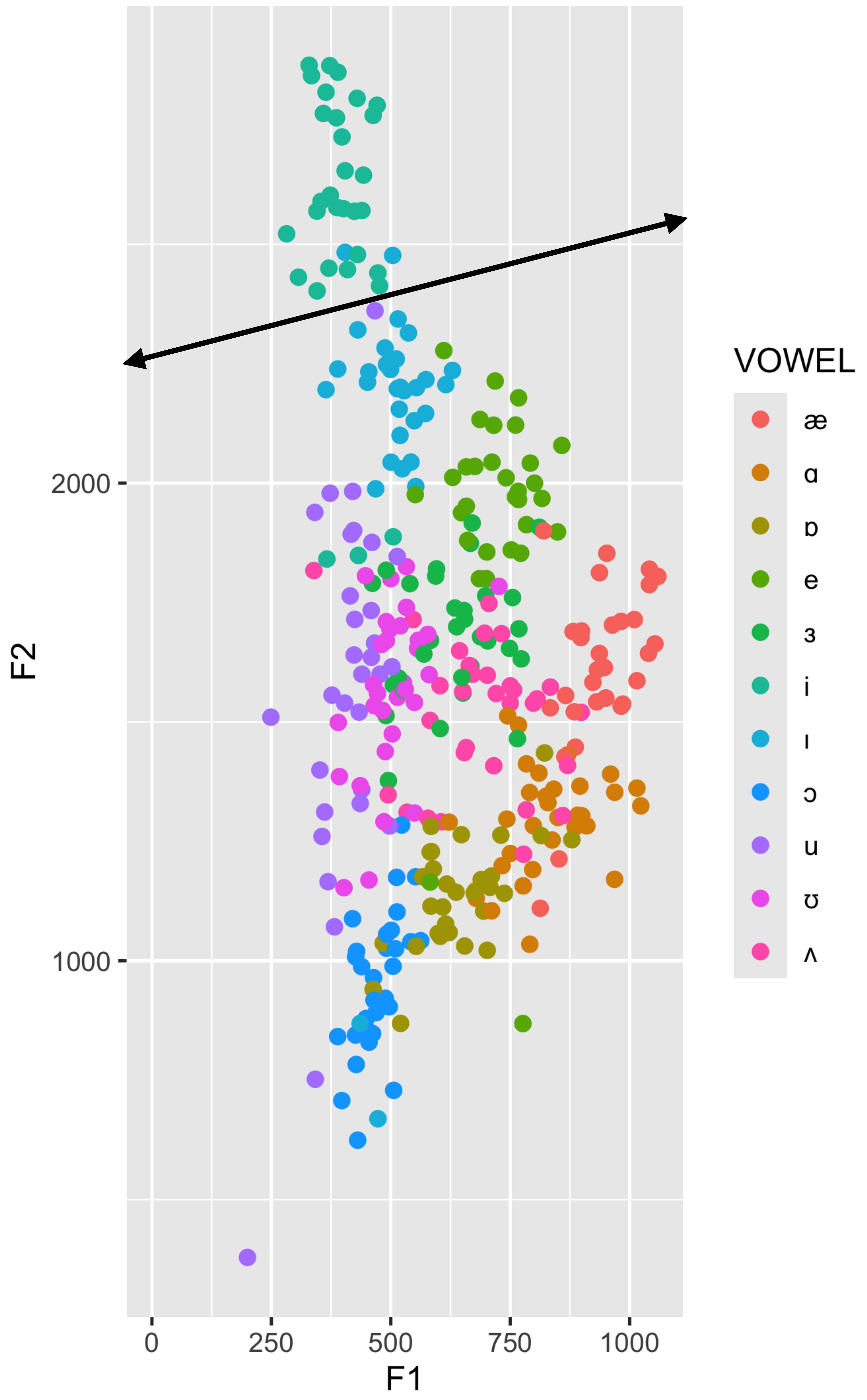
- We'd probably end up with something like this drawn line
- Line formula: $y = mx + b$
  - $\text{F2} = m \cdot \text{F1} + b$
  - $b \approx 2265$ (y-intercept)
  - $m \approx 294/1125$ (slope, "rise over run")
  - $\text{F2} = \frac{294}{1125}\text{F1} + 2265$
- Let's rearrange this:

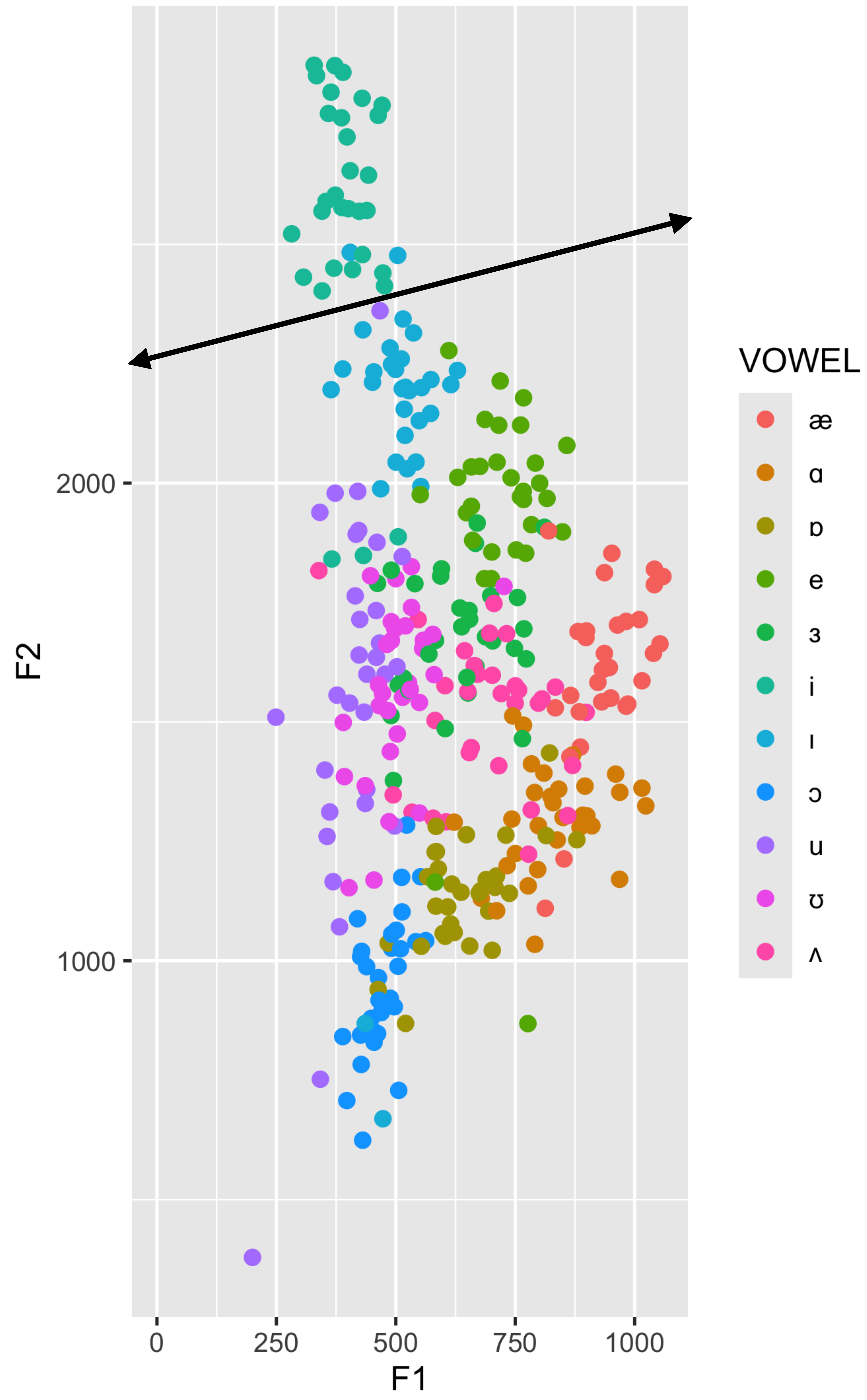
$$\text{F2} - \frac{294}{1125}\text{F1} - 2265 = 0$$

$$1.0 \cdot \text{F2} + (-0.26) \cdot \text{F1} - 2265 = 0$$

$$\boxed{-0.26 \cdot \text{F1} + 1.0 \cdot \text{F2} - 2265 = 0}$$

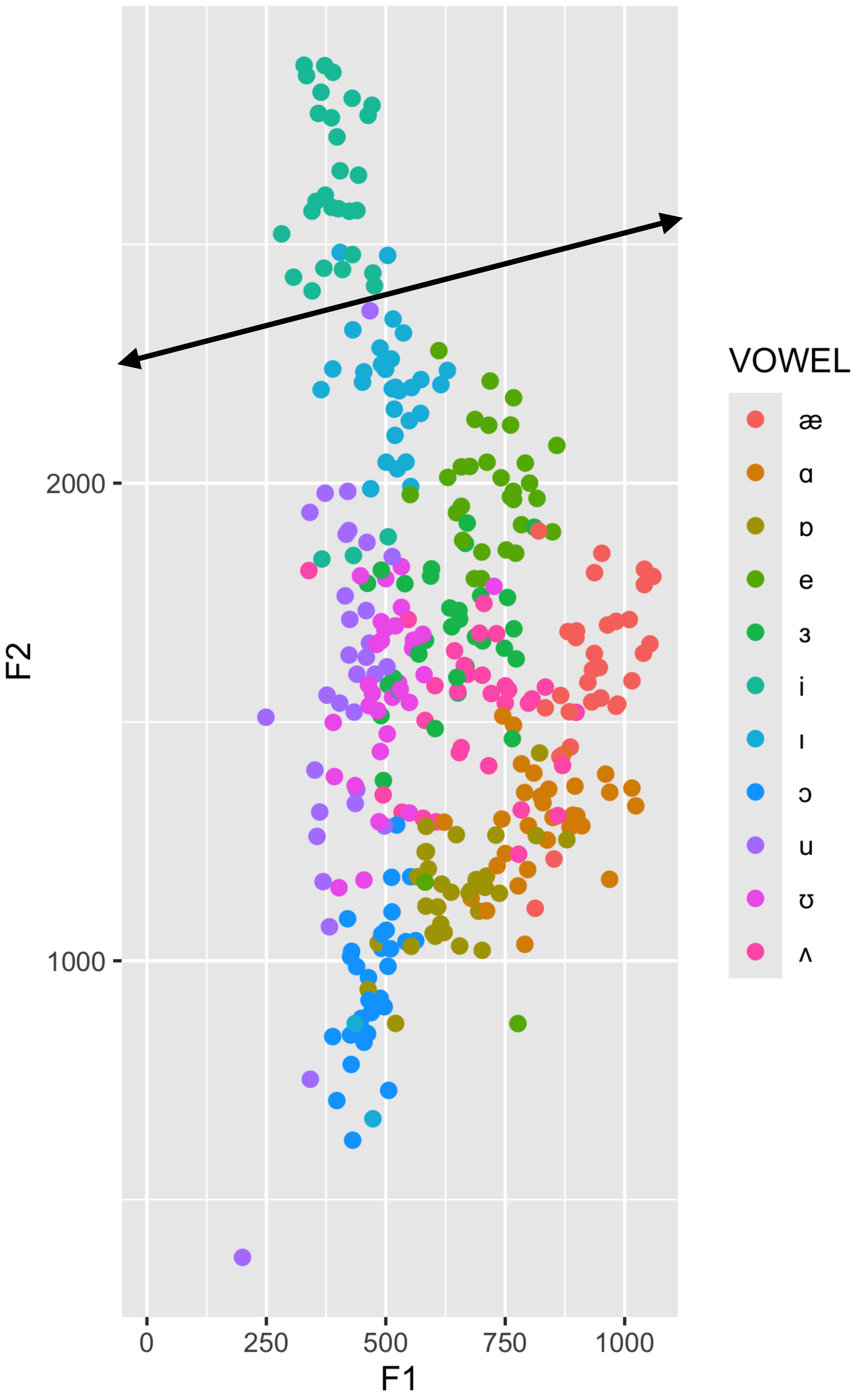
What does this remind us of?



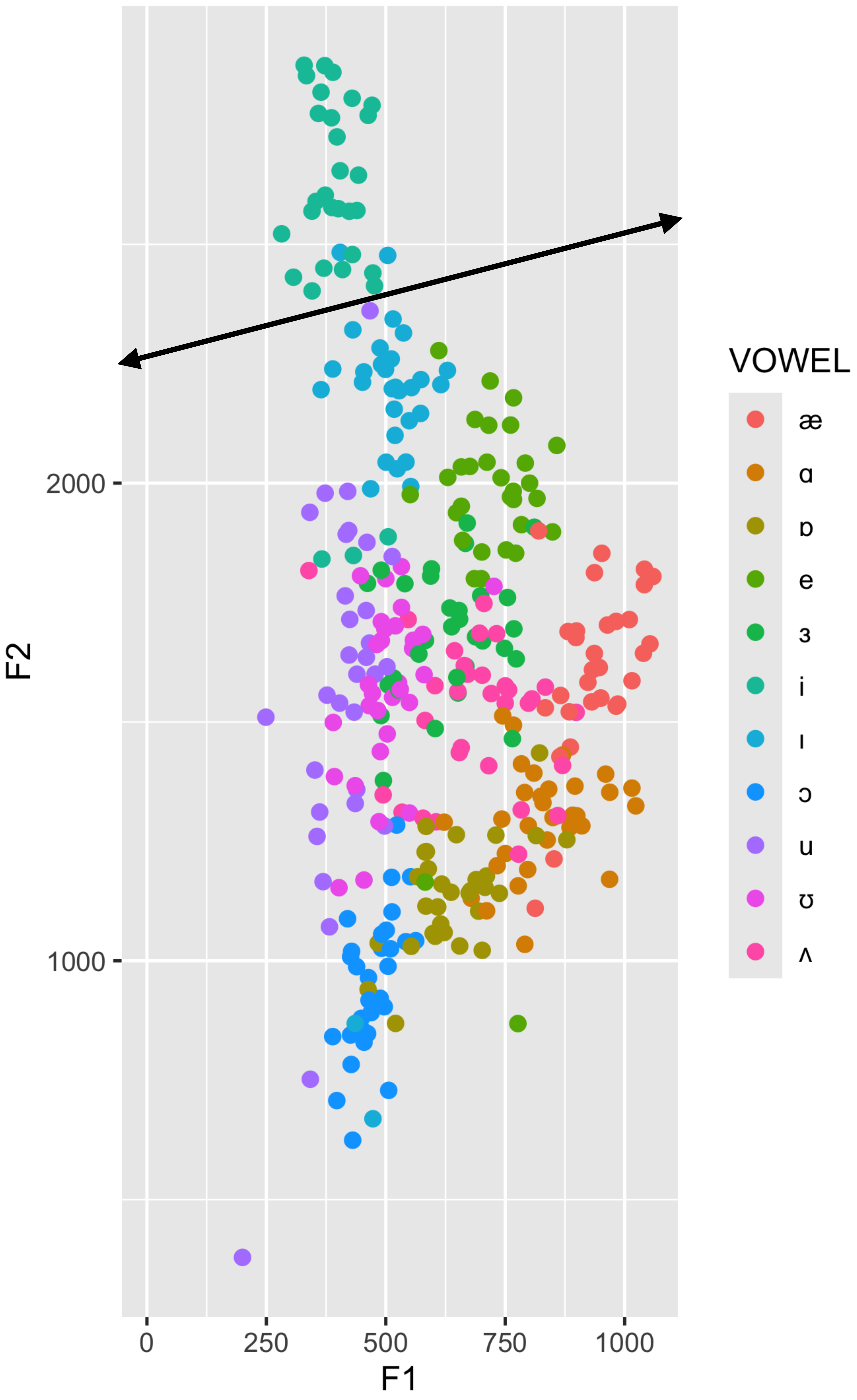


VOWEL
æ
ɑ
ɒ
e
ɜ
i
ɪ
ɔ
u
ʊ
ʌ
2000
1000
F2
0
250
500
750
1000
F1

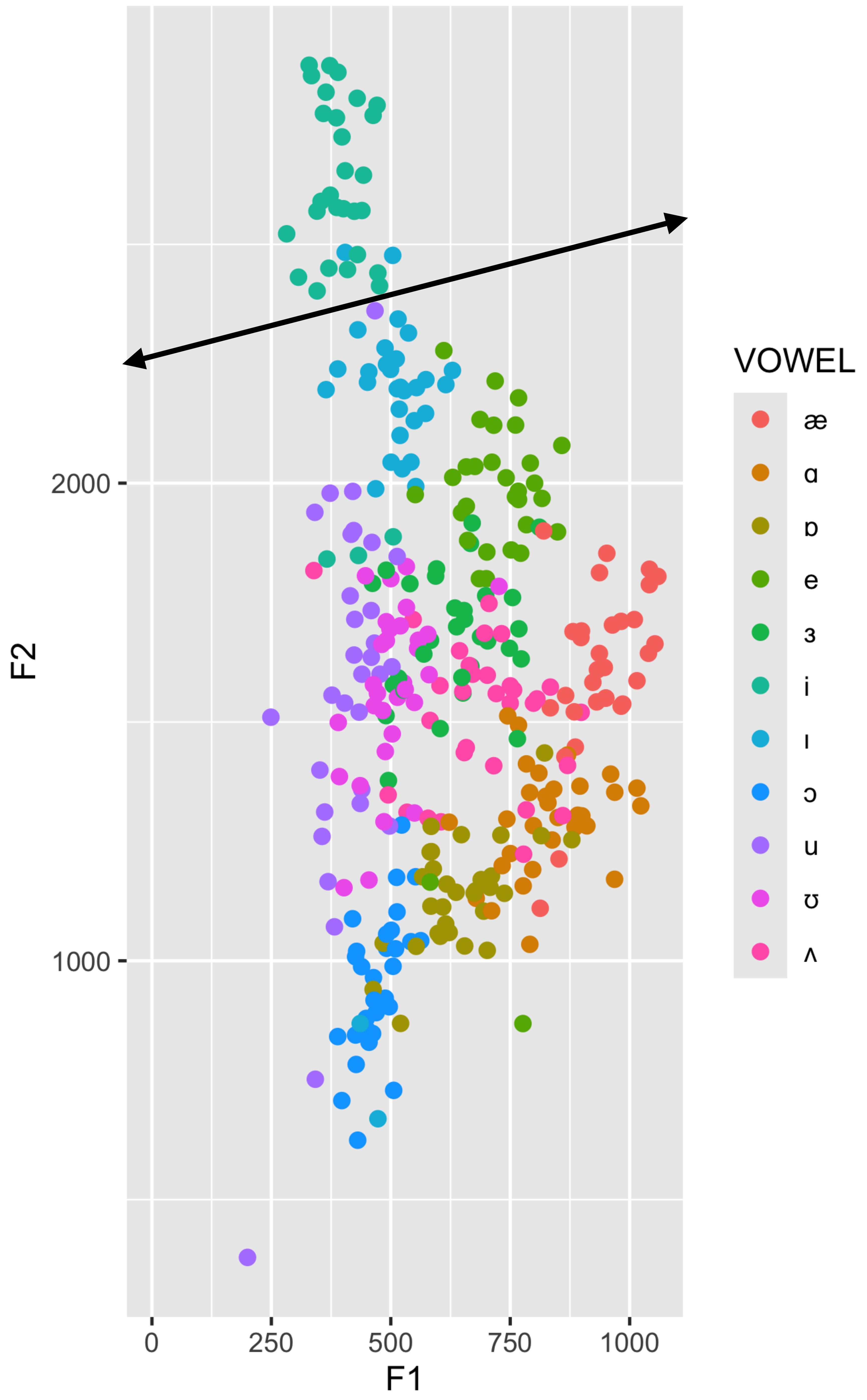
- Our equation for the **boundary**:



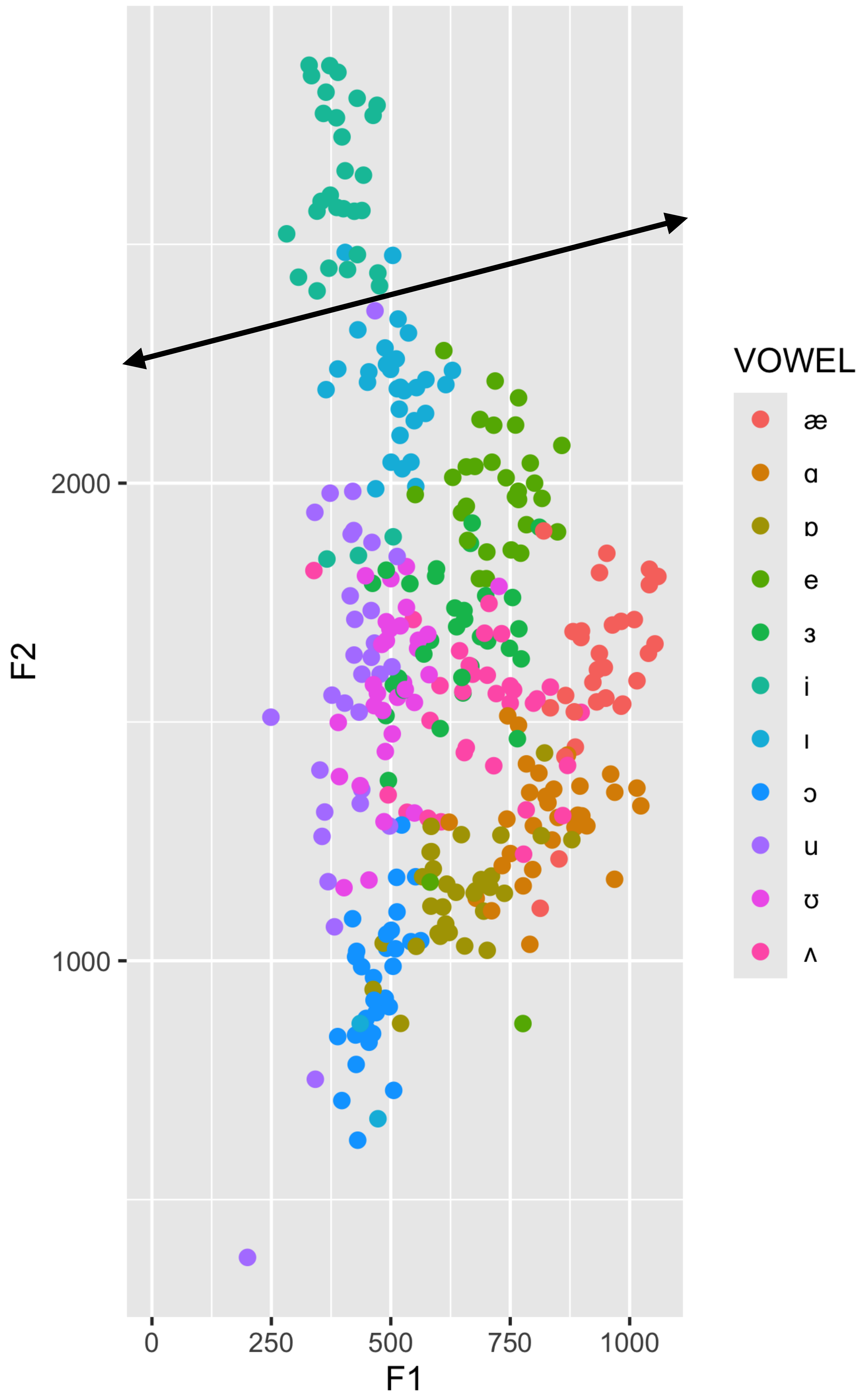
- Our equation for the **boundary**:
  - $-0.26 \cdot \text{F1} + 1.0 \cdot \text{F2} - 2265$



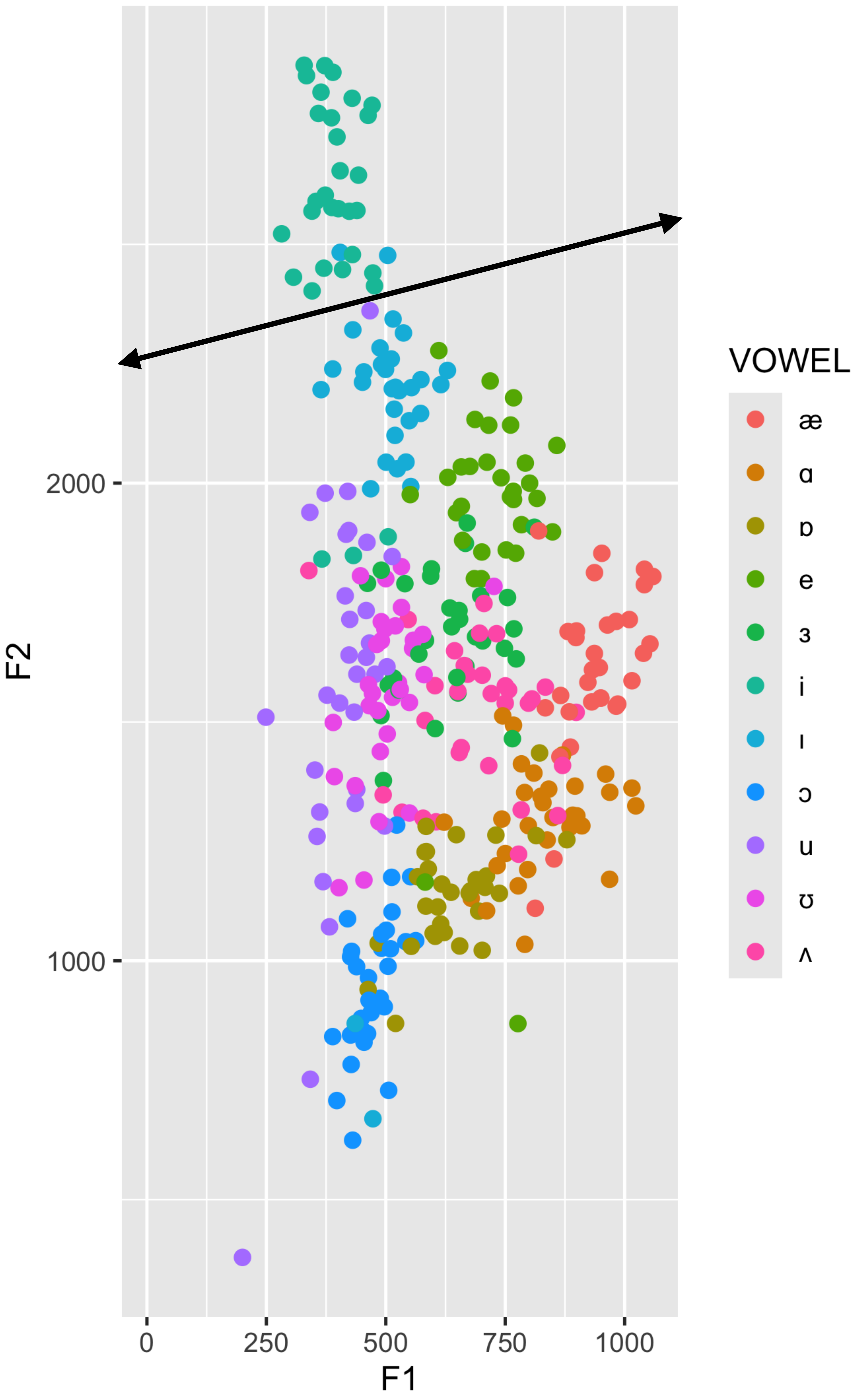
- Our equation for the **boundary**:
  - $-0.26 \cdot \text{F1} + 1.0 \cdot \text{F2} - 2265$
  - fits the form $w_1 x_1 + w_2 x_2 + b$



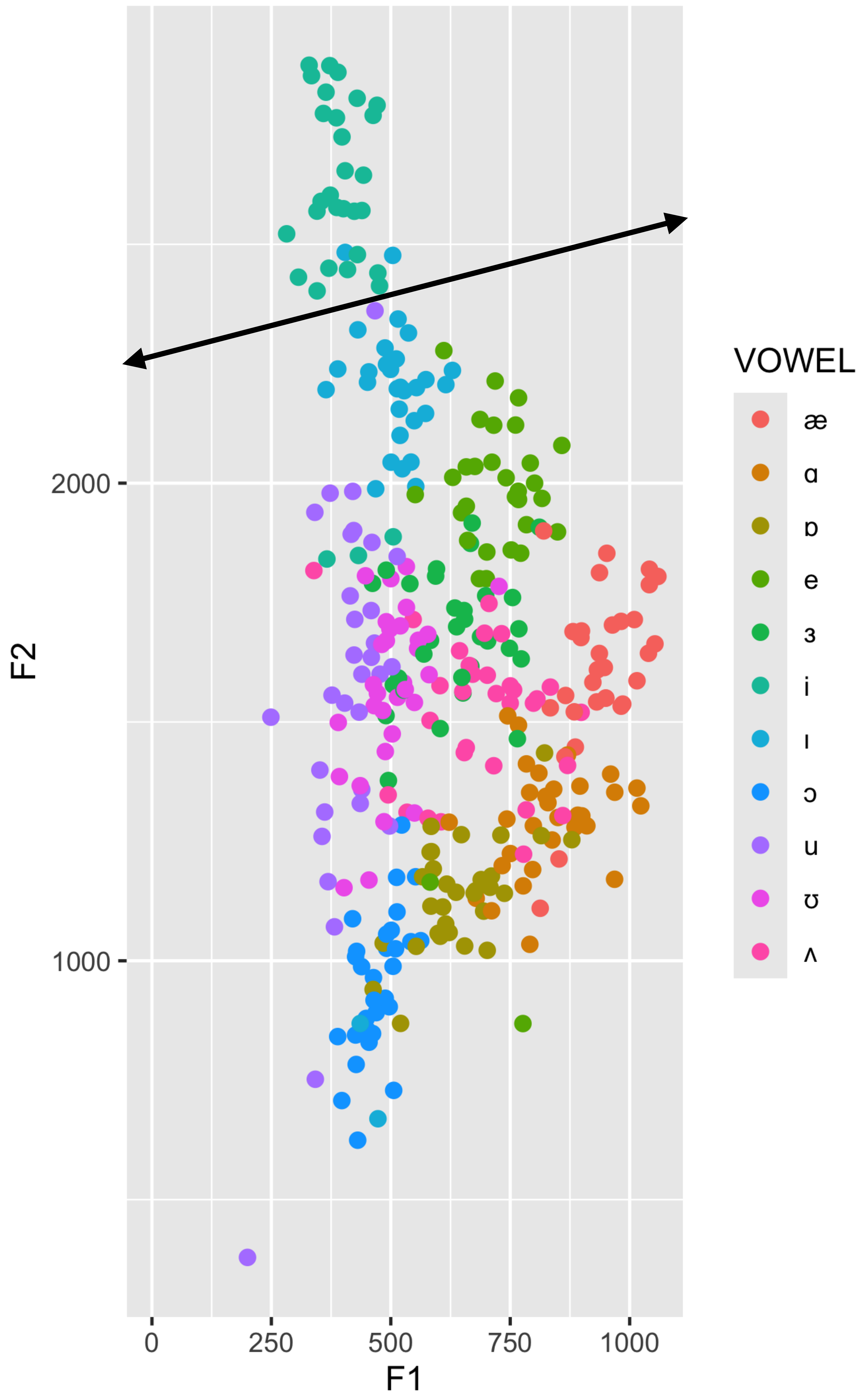
- Our equation for the **boundary**:
  - $-0.26 \cdot \text{F1} + 1.0 \cdot \text{F2} - 2265$
  - fits the form $w_1x_1 + w_2x_2 + b$
  - $(w \cdot x + b)$!



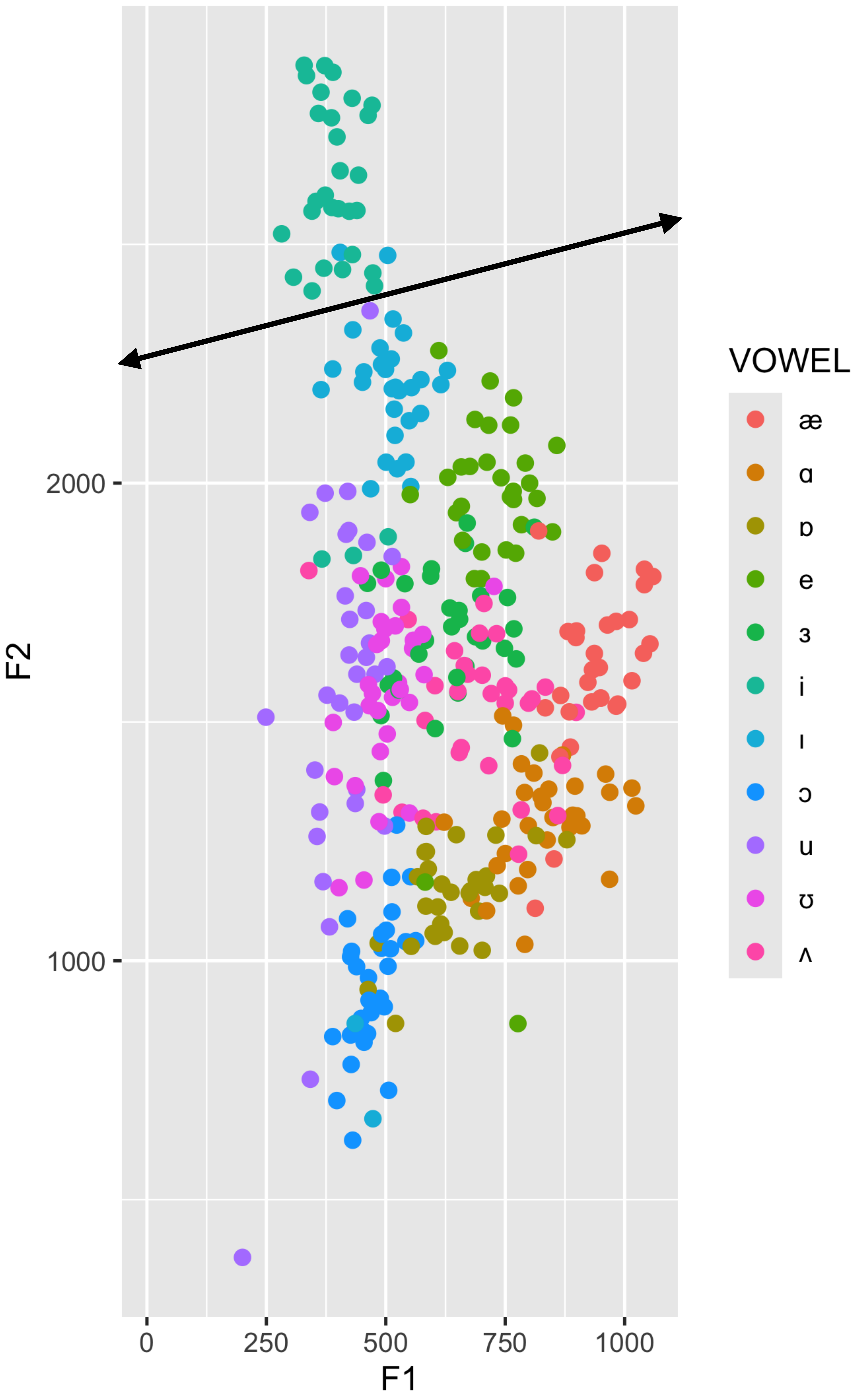
- Our equation for the **boundary**:
  - $-0.26 \cdot \text{F1} + 1.0 \cdot \text{F2} - 2265$
  - fits the form $w_1 x_1 + w_2 x_2 + b$
  - $(w \cdot x + b)$!
- What does the **Perceptron** $\sigma(w \cdot x + b)$ do?



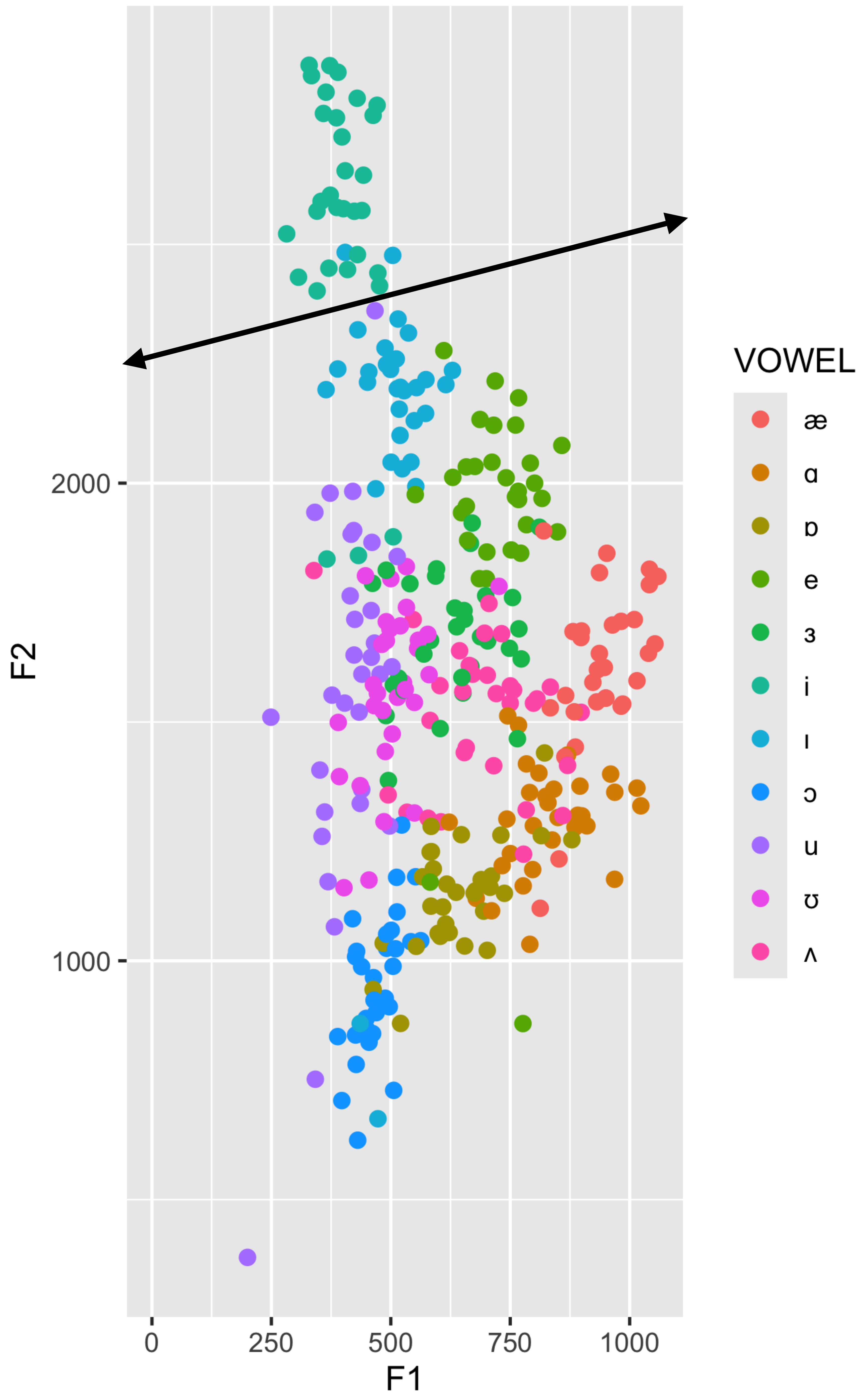
- Our equation for the **boundary**:
  - $-0.26 \cdot \mathsf{F1} + 1.0 \cdot \mathsf{F2} - 2265$
  - fits the form $w_1 x_1 + w_2 x_2 + b$
  - $(w \cdot x + b)$!
- What does the **Perceptron** $\sigma(w \cdot x + b)$ do?
  - **Hint:** what happens when you...



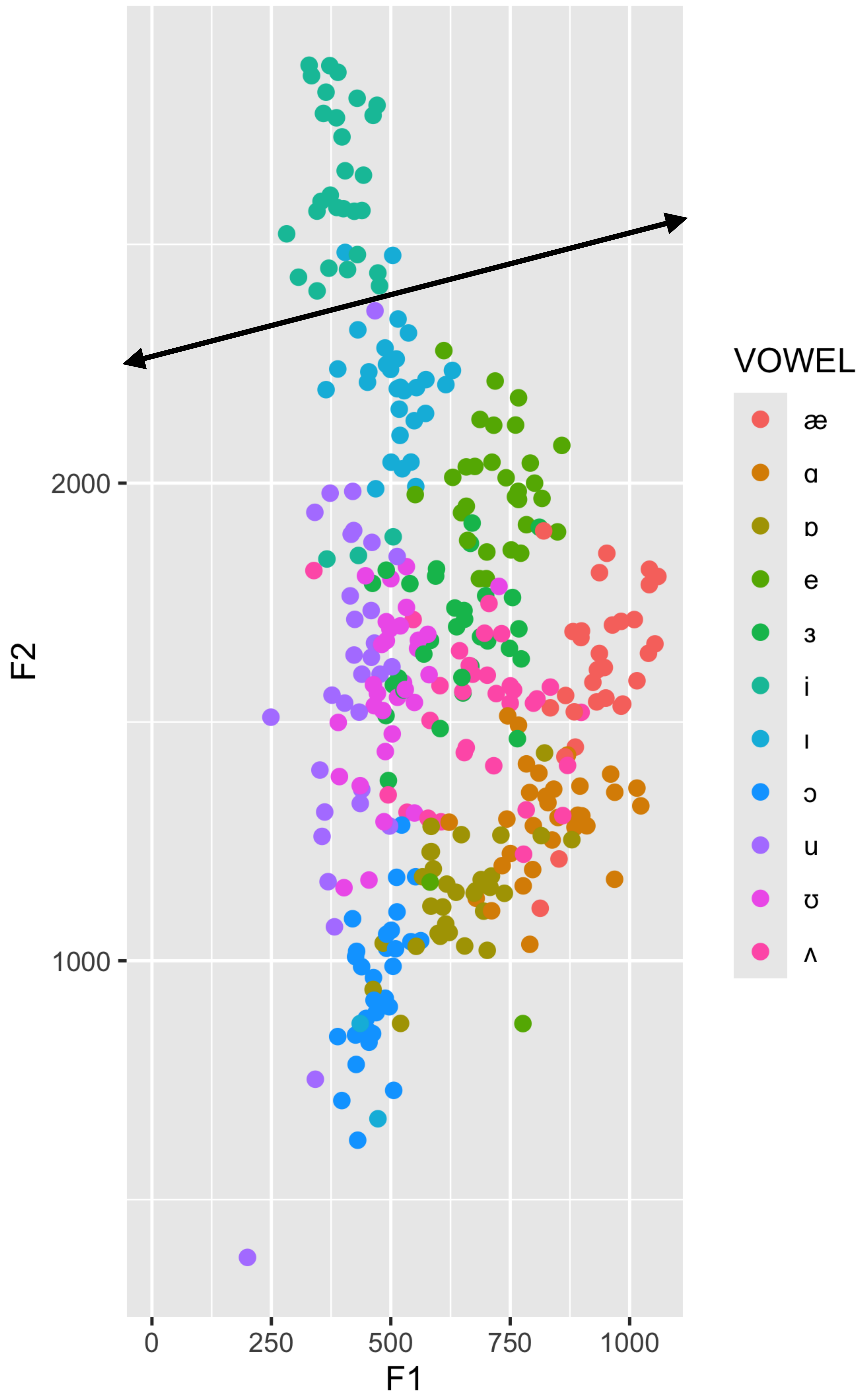
- Our equation for the **boundary**:
  - $-0.26 \cdot \text{F1} + 1.0 \cdot \text{F2} - 2265$
  - fits the form $w_1 x_1 + w_2 x_2 + b$
  - $(w \cdot x + b)$!
- What does the **Perceptron** $\sigma(w \cdot x + b)$ do?
  - **Hint:** what happens when you...
    - ...start **on the boundary**...



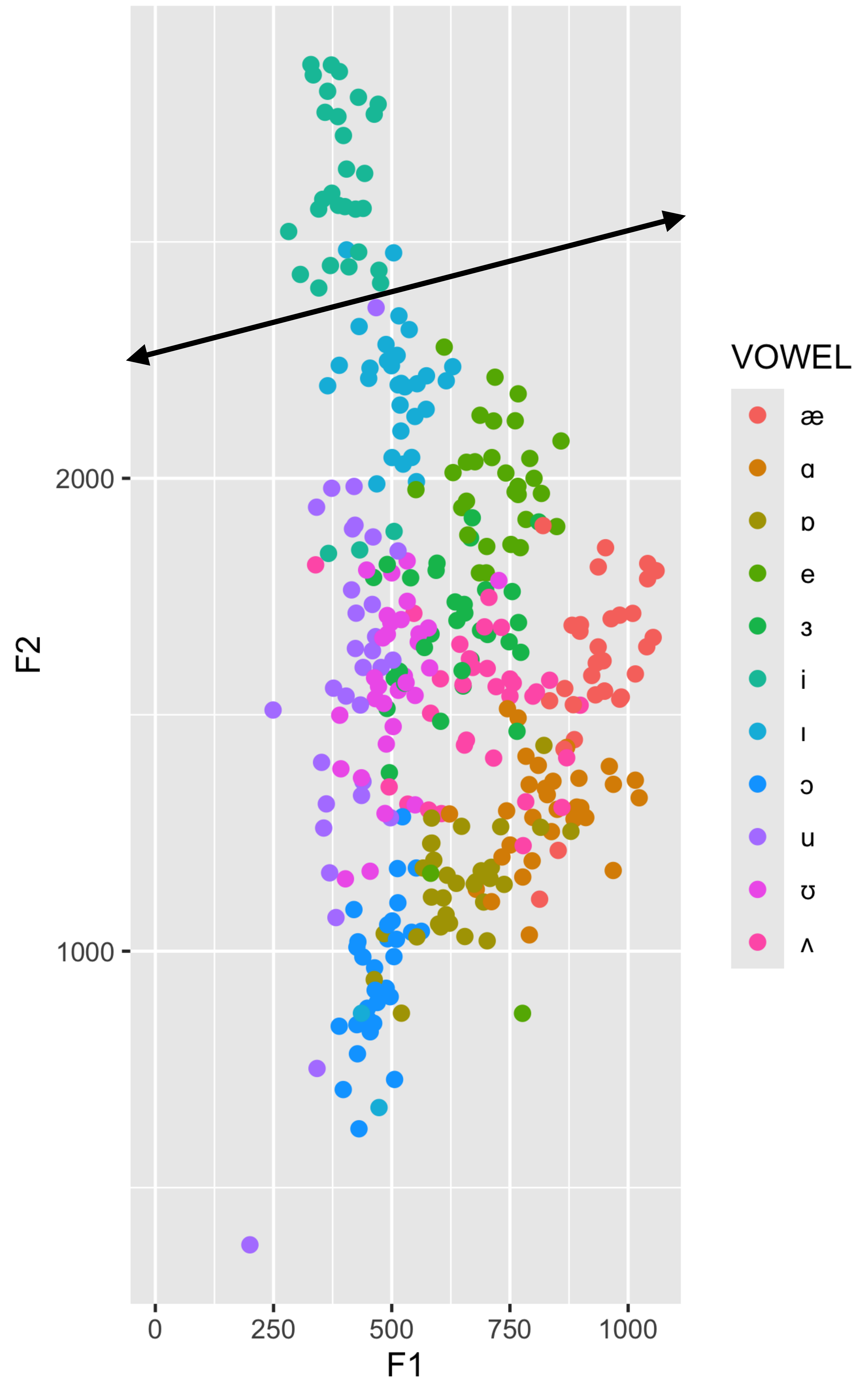
- Our equation for the **boundary**:
  - $-0.26 \cdot \text{F1} + 1.0 \cdot \text{F2} - 2265$
  - fits the form $w_1 x_1 + w_2 x_2 + b$
  - $(w \cdot x + b)$!
- What does the **Perceptron** $\sigma(w \cdot x + b)$ do?
  - **Hint:** what happens when you...
    - ...start **on the boundary**...
    - ...and **decrease F1** or **increase F2?**



- Our equation for the **boundary**:
  - $-0.26 \cdot \text{F1} + 1.0 \cdot \text{F2} - 2265$
  - fits the form $w_1x_1 + w_2x_2 + b$
  - $(w \cdot x + b)$!
- What does the **Perceptron** $\sigma(w \cdot x + b)$ do?
  - **Hint:** what happens when you...
    - ...start **on the boundary**...
    - ...and **decrease F1** or **increase F2?**
  - As $wx + b$ becomes **more positive**, $\sigma(w \cdot x + b)$ **approaches 1.0!**



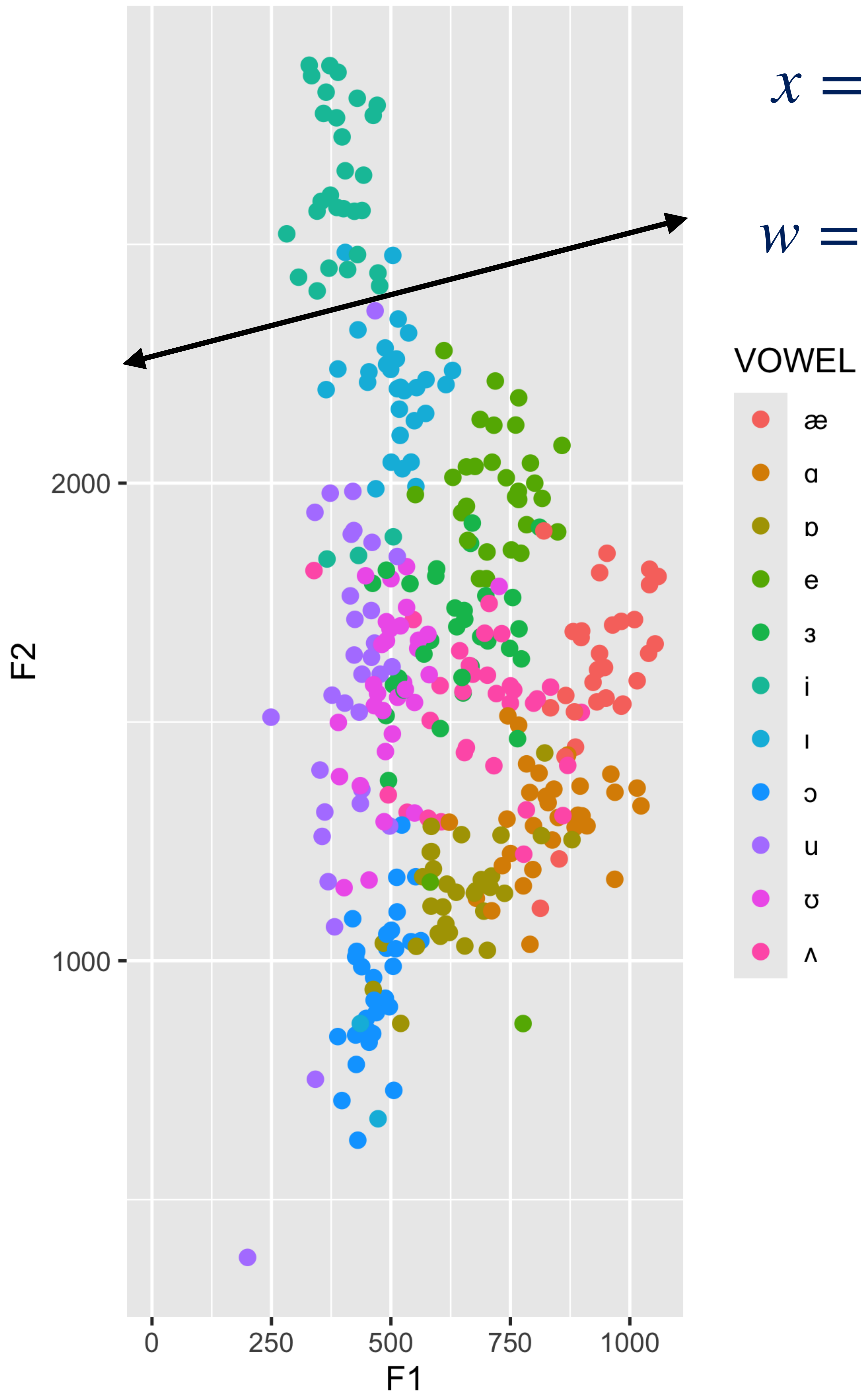
- Our equation for the **boundary**:
  - $-0.26 \cdot \text{F1} + 1.0 \cdot \text{F2} - 2265$
  - fits the form $w_1x_1 + w_2x_2 + b$
  - $(w \cdot x + b)$!
- What does the **Perceptron** $\sigma(w \cdot x + b)$ do?
  - **Hint:** what happens when you...
    - ...start **on the boundary**...
    - ...and **decrease F1** or **increase F2?**
  - As $wx + b$ becomes **more positive**, $\sigma(w \cdot x + b)$ **approaches 1.0!**
  - I.e. the Perceptron becomes **certain of a positive classification of [i]**



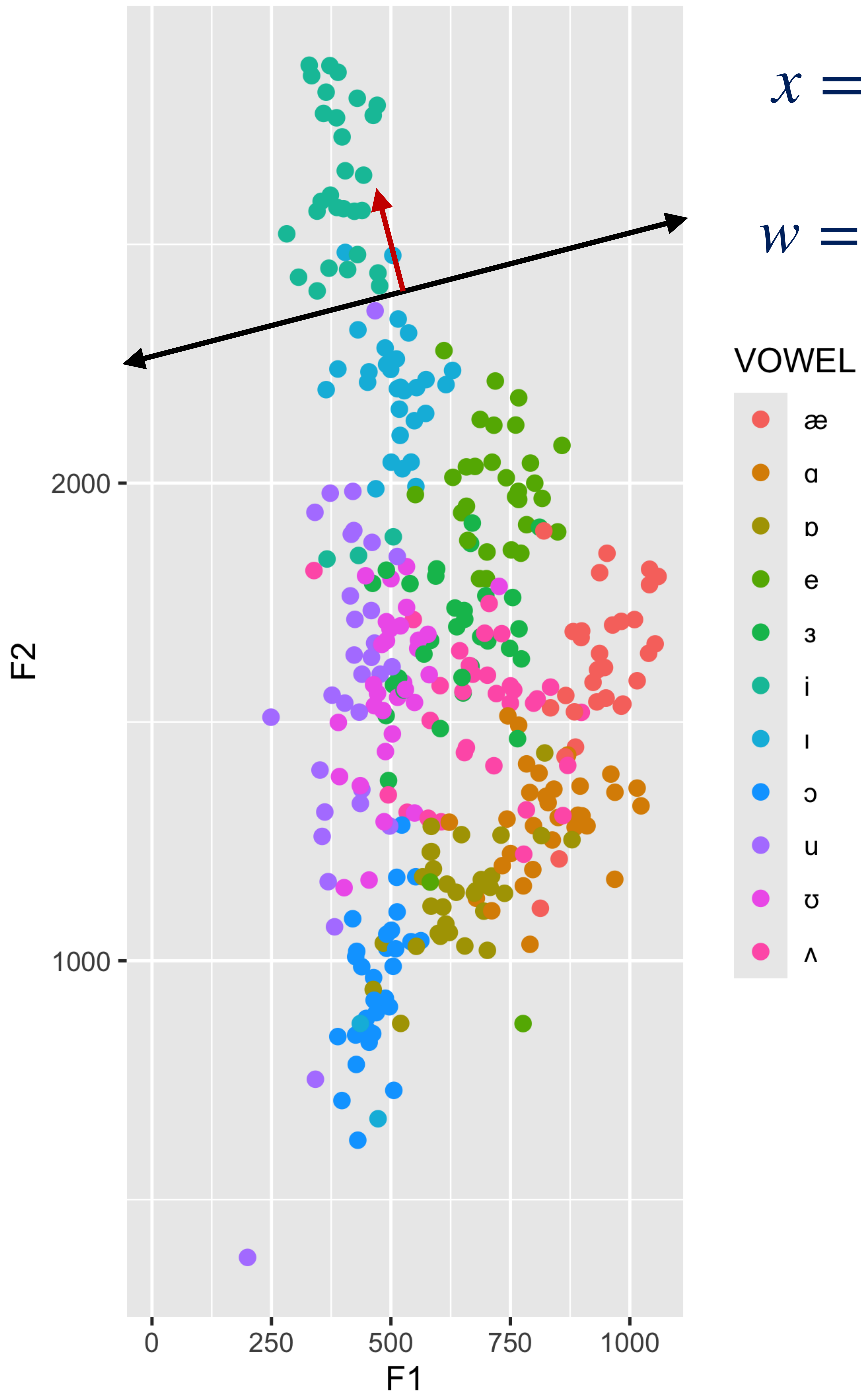
- Our equation for the **boundary**:
  - $-0.26 \cdot \text{F1} + 1.0 \cdot \text{F2} - 2265$
  - fits the form $w_1x_1 + w_2x_2 + b$
  - $(w \cdot x + b)$!
- What does the **Perceptron** $\sigma(w \cdot x + b)$ do?
  - **Hint:** what happens when you...
    - ...start **on the boundary**...
    - ...and **decrease F1** or **increase F2?**
  - As $wx + b$ becomes **more positive**, $\sigma(w \cdot x + b)$ **approaches 1.0!**
  - I.e. the Perceptron becomes **certain of a positive classification of [i]**

$$x = \begin{bmatrix} \text{F1} \\ \text{F2} \end{bmatrix}$$

$$w = \begin{bmatrix} \text{-0.26} \\ 1.0 \end{bmatrix}$$



- Our equation for the **boundary**:
  - $-0.26 \cdot \text{F1} + 1.0 \cdot \text{F2} - 2265$
  - fits the form $w_1 x_1 + w_2 x_2 + b$
  - $(w \cdot x + b)$!
- What does the **Perceptron** $\sigma(w \cdot x + b)$ do?
  - **Hint:** what happens when you...
    - ...start **on the boundary**...
    - ...and **decrease F1** or **increase F2?**
  - As $wx + b$ becomes **more positive**, $\sigma(w \cdot x + b)$ **approaches 1.0!**
  - I.e. the Perceptron becomes **certain of a positive classification of [i]**



$$x = \begin{bmatrix} \text{F1} \\ \text{F2} \end{bmatrix}$$

$$w = \begin{bmatrix} -0.26 \\ 1.0 \end{bmatrix}$$

Logical Functions

p	q	p∧q	p∨q
0	0	0	0
1	0	0	1
0	1	0	1
1	1	1	1

Logical Functions

- Logical (or "Boolean") functions give **truth values** based compositionally on the truth value of their inputs

p	q	$p \wedge q$	$p \vee q$
0	0	0	0
1	0	0	1
0	1	0	1
1	1	1	1

Logical Functions

- Logical (or "Boolean") functions give **truth values** based compositionally on the truth value of their inputs
- This **truth table** gives the values of **logical AND (∧)** and **logical OR (∨)** based on some input variables p and q

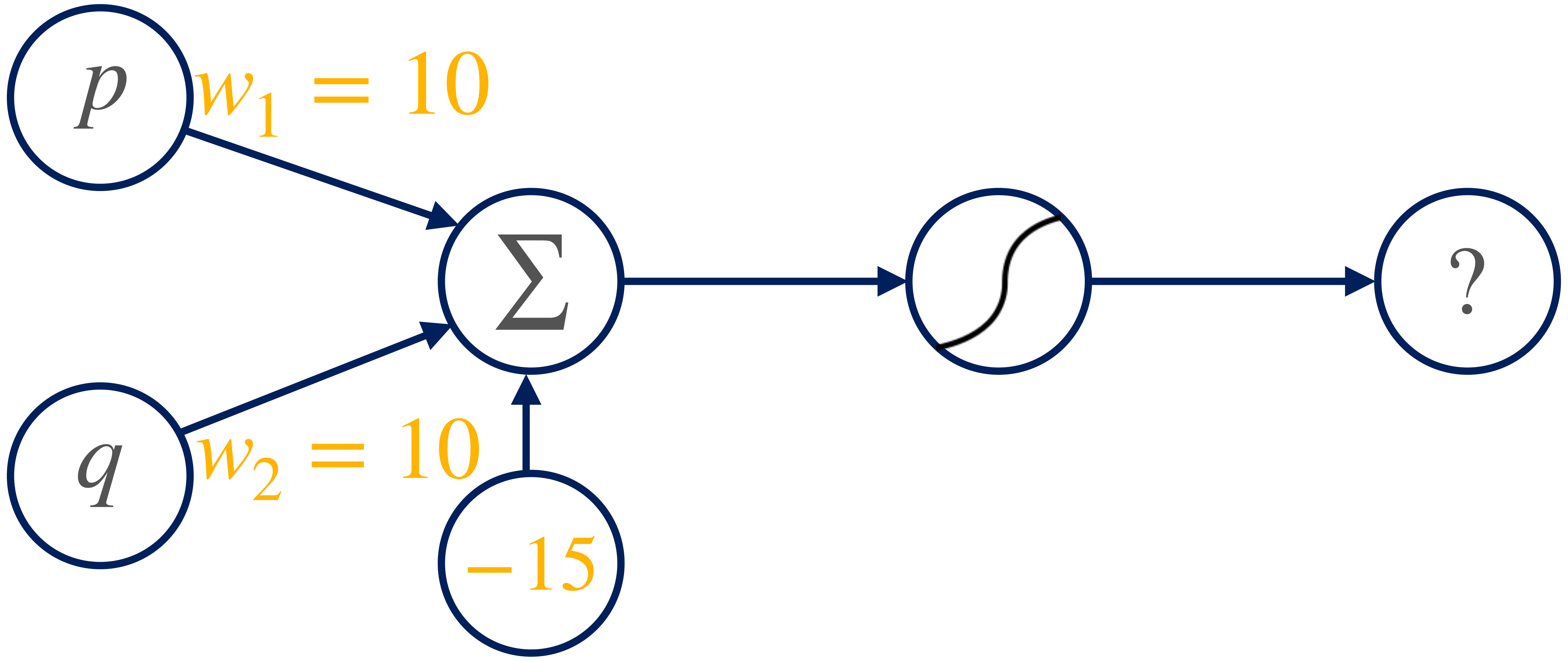
p	q	p∧q	p∨q
0	0	0	0
1	0	0	1
0	1	0	1
1	1	1	1

Logical Functions

- Logical (or "Boolean") functions give **truth values** based compositionally on the truth value of their inputs
- This **truth table** gives the values of **logical AND (∧)** and **logical OR (∨)** based on some input variables p and q
- **"Boolean logic"** is pervasive in many fields such as **Semantics, Computer Science, and Electrical Engineering**

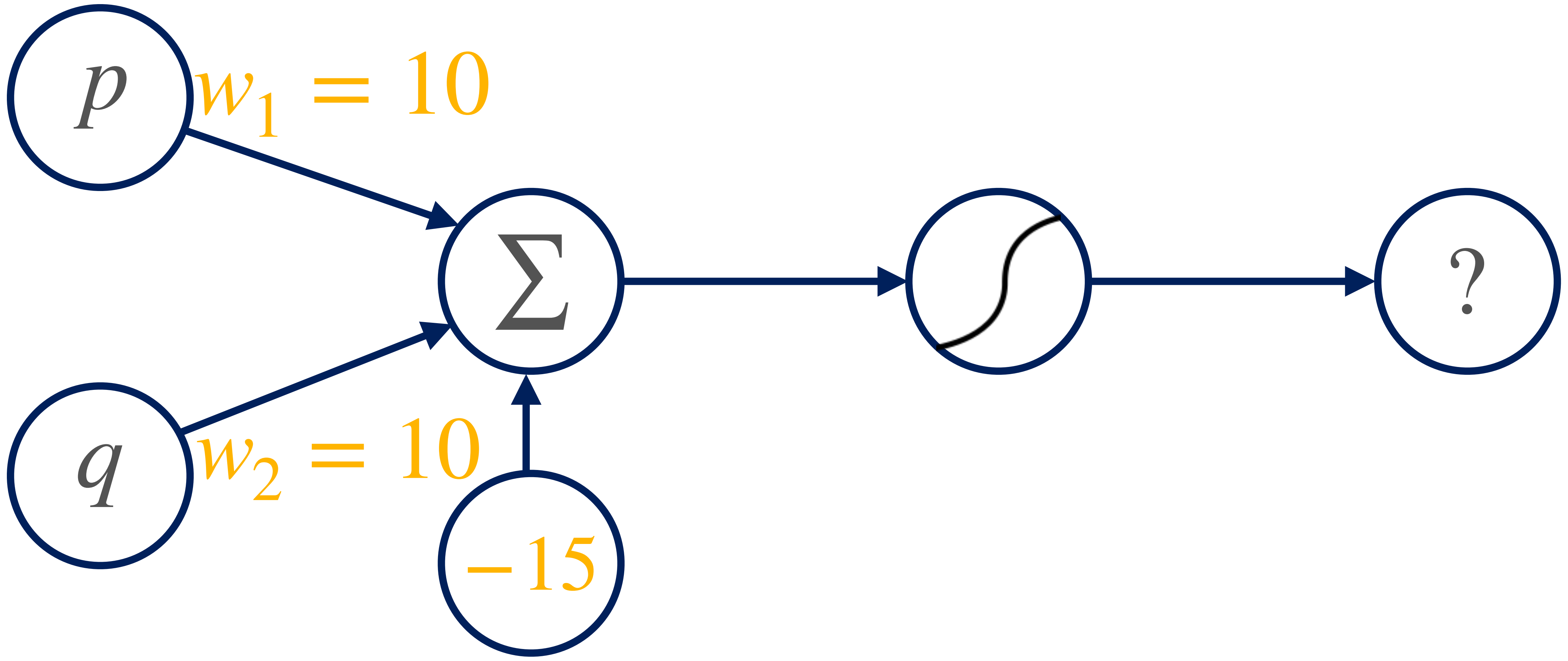
p	q	p∧q	p∨q
0	0	0	0
1	0	0	1
0	1	0	1
1	1	1	1

AND Perceptron



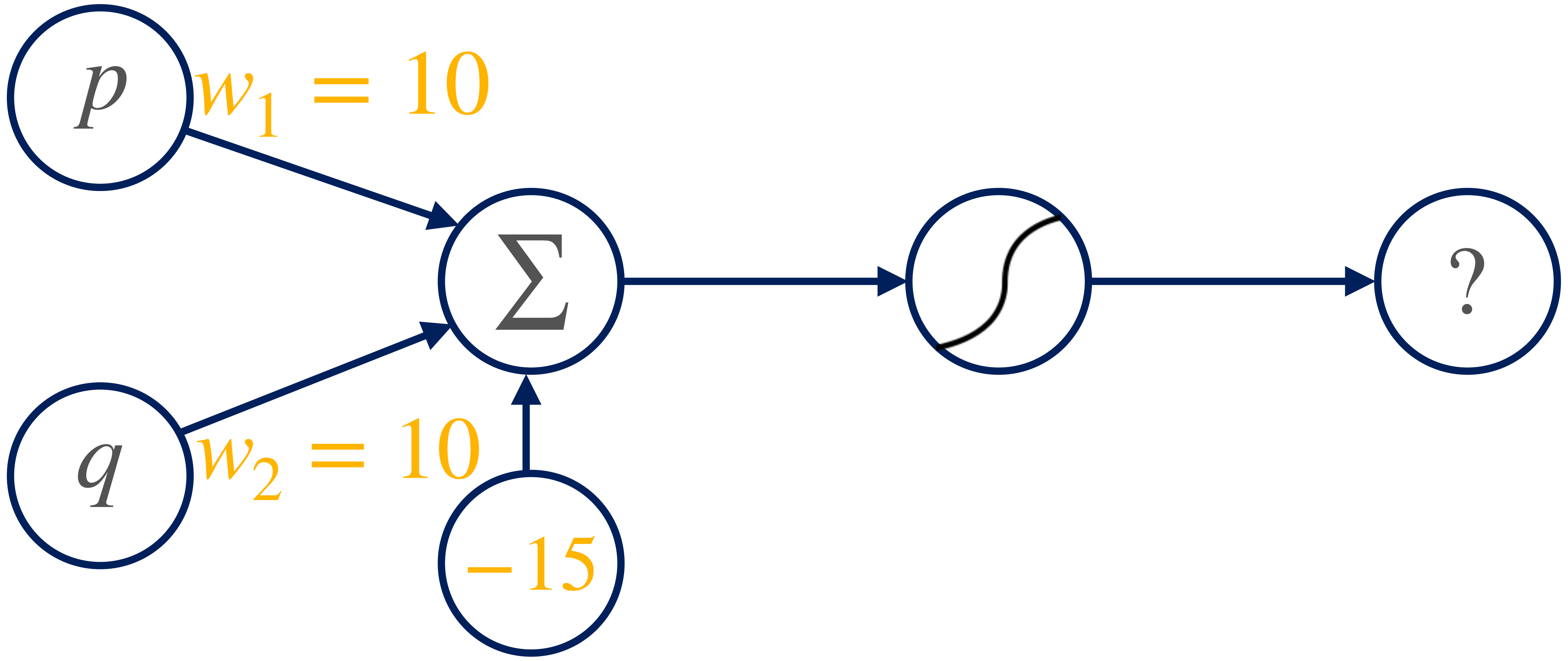
AND Perceptron

- Assume that p and q can ONLY take on the values {0,1}



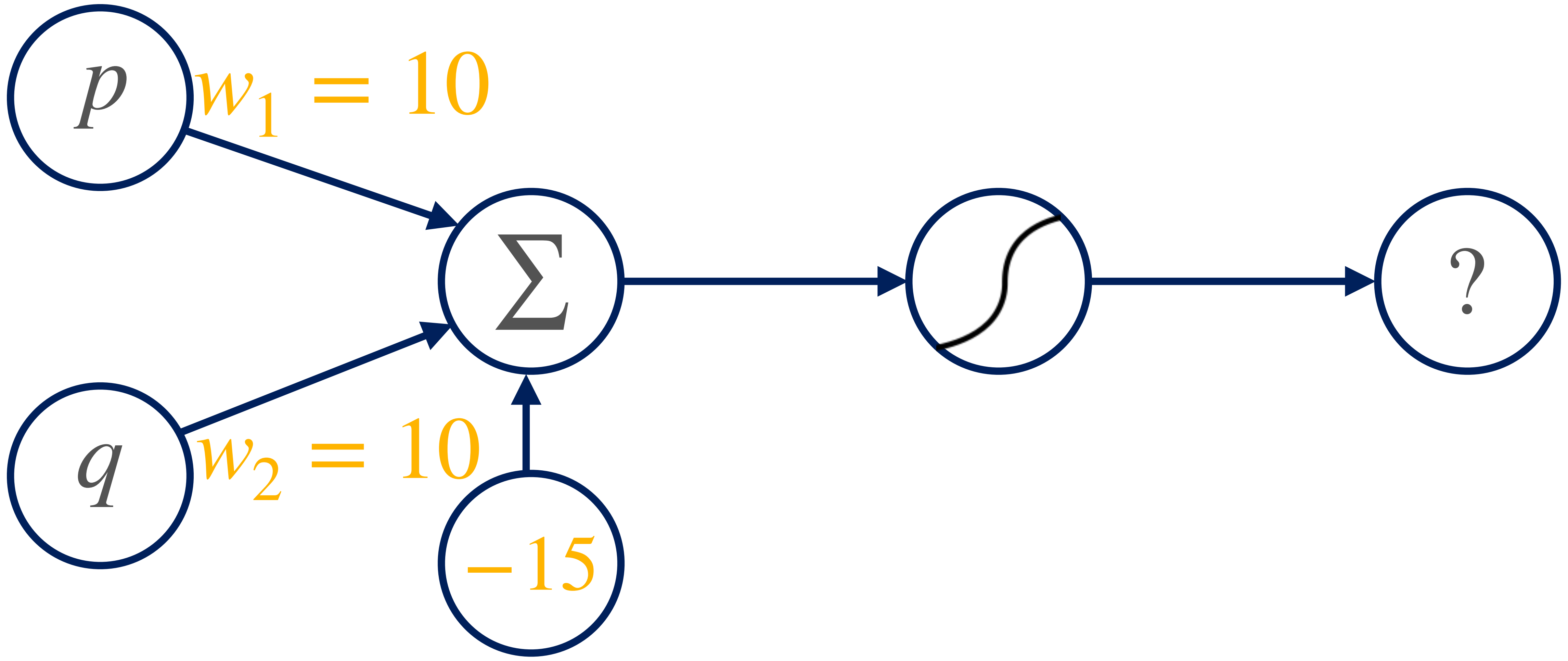
AND Perceptron

- Assume that p and q can ONLY take on the values {0,1}
- When is Perceptron output **close to 1.0?** What is $wx + b$?



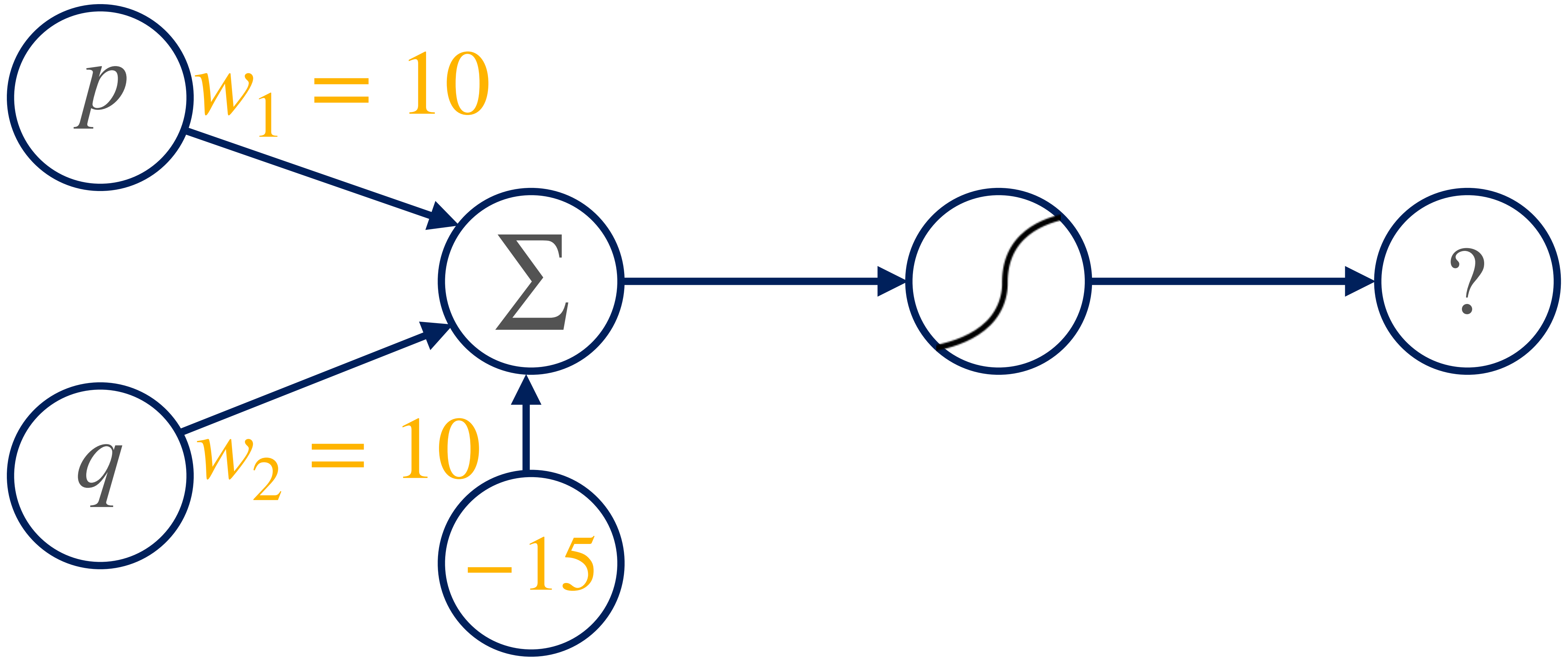
AND Perceptron

- Assume that p and q can ONLY take on the values {0,1}
- When is Perceptron output **close to 1.0?** What is $wx + b$?
  - When $wx + b$ is **positive**, i.e. when **both p and q** are 1



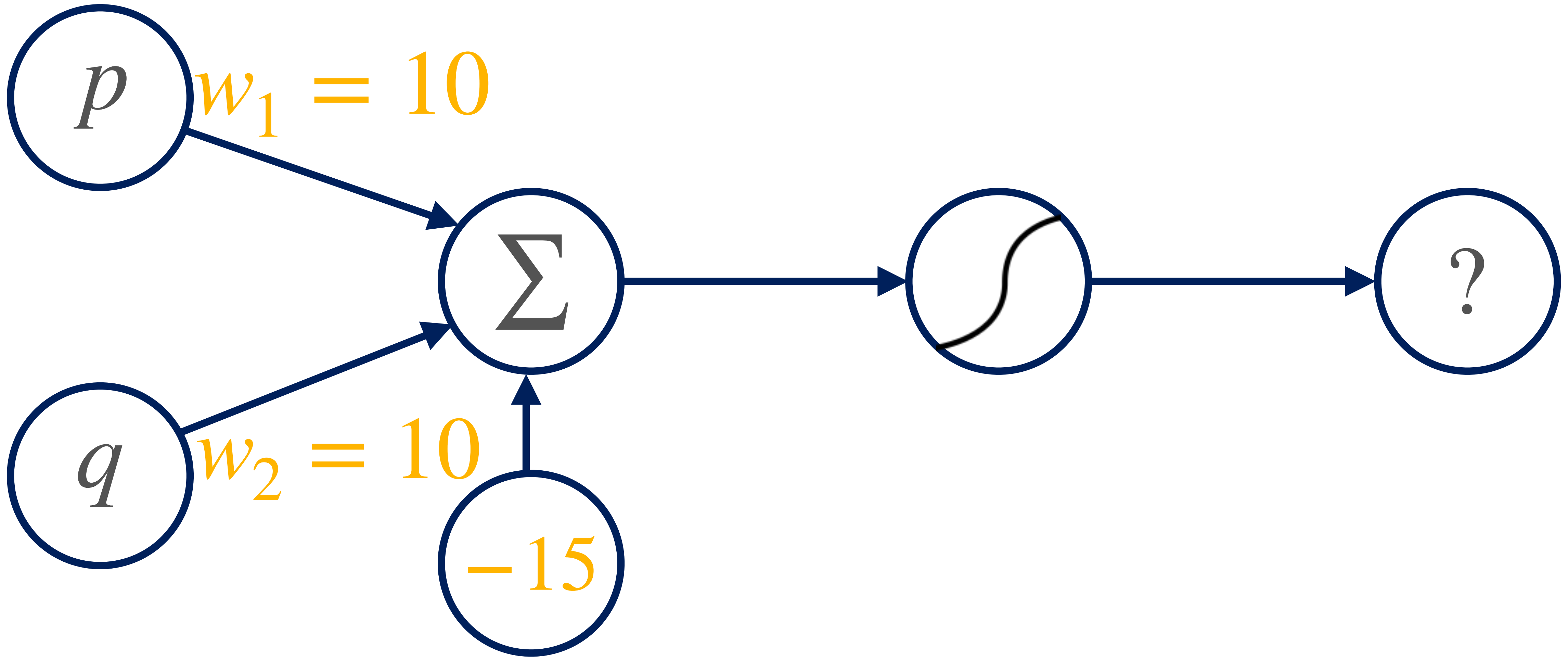
AND Perceptron

- Assume that p and q can ONLY take on the values {0,1}
- When is Perceptron output **close to 1.0?** What is $wx + b$?
  - When $wx + b$ is **positive**, i.e. when **both p and q** are 1
- When is the output **close to 0.0?** What is $wx + b$?

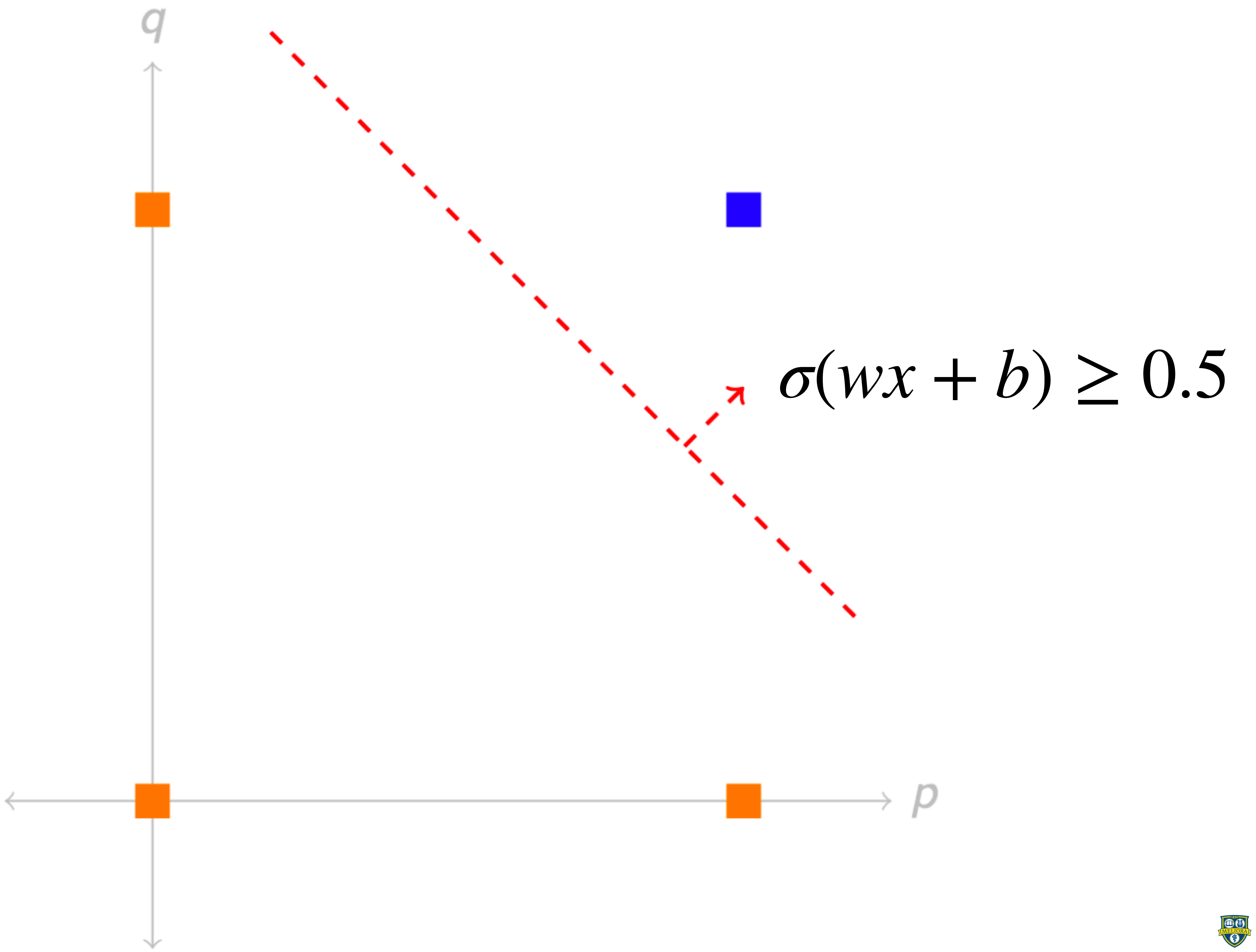


AND Perceptron

- Assume that p and q can ONLY take on the values {0,1}
- When is Perceptron output **close to 1.0?** What is $wx + b$?
  - When $wx + b$ is **positive**, i.e. when **both p and q** are 1
- When is the output **close to 0.0?** What is $wx + b$?
  - When $wx + b$ is **negative**, i.e. **either p or q is 0**

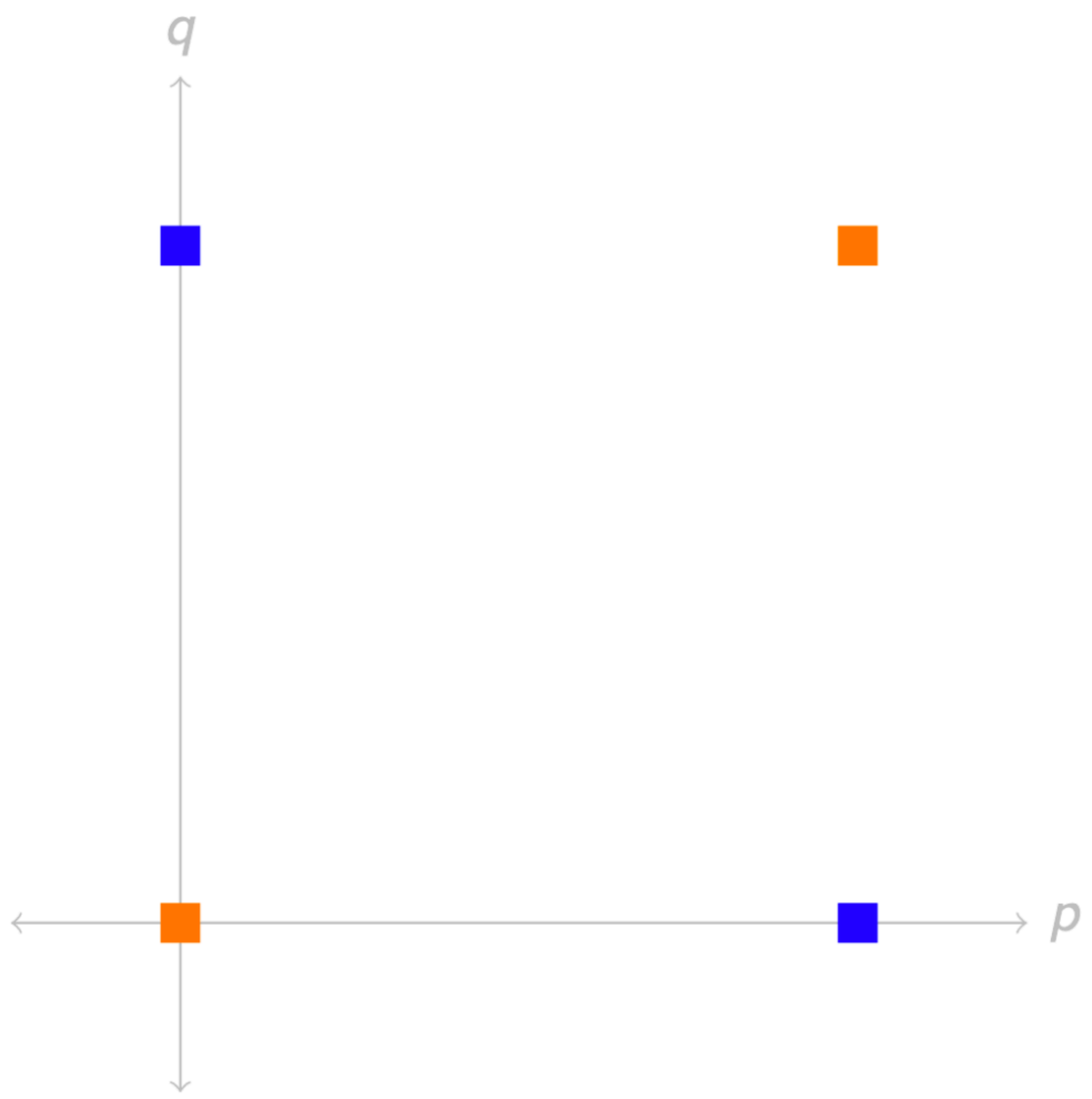


AND Linear Separation



XOR Function

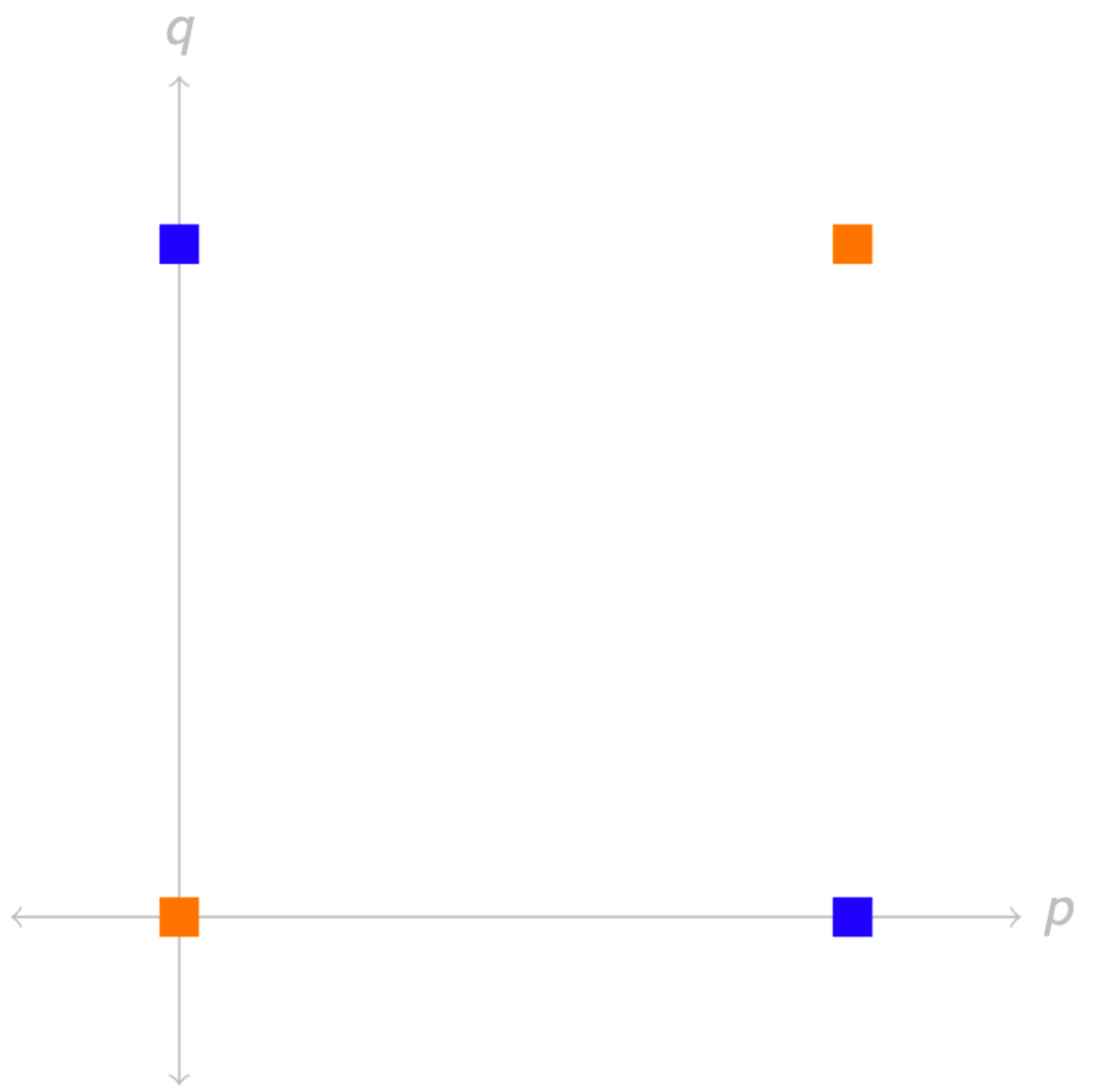
p	q	p XOR q
0	0	0
1	0	1
0	1	1
1	1	0



XOR Function

- The XOR function is True when **either** p or q are True **but not both**

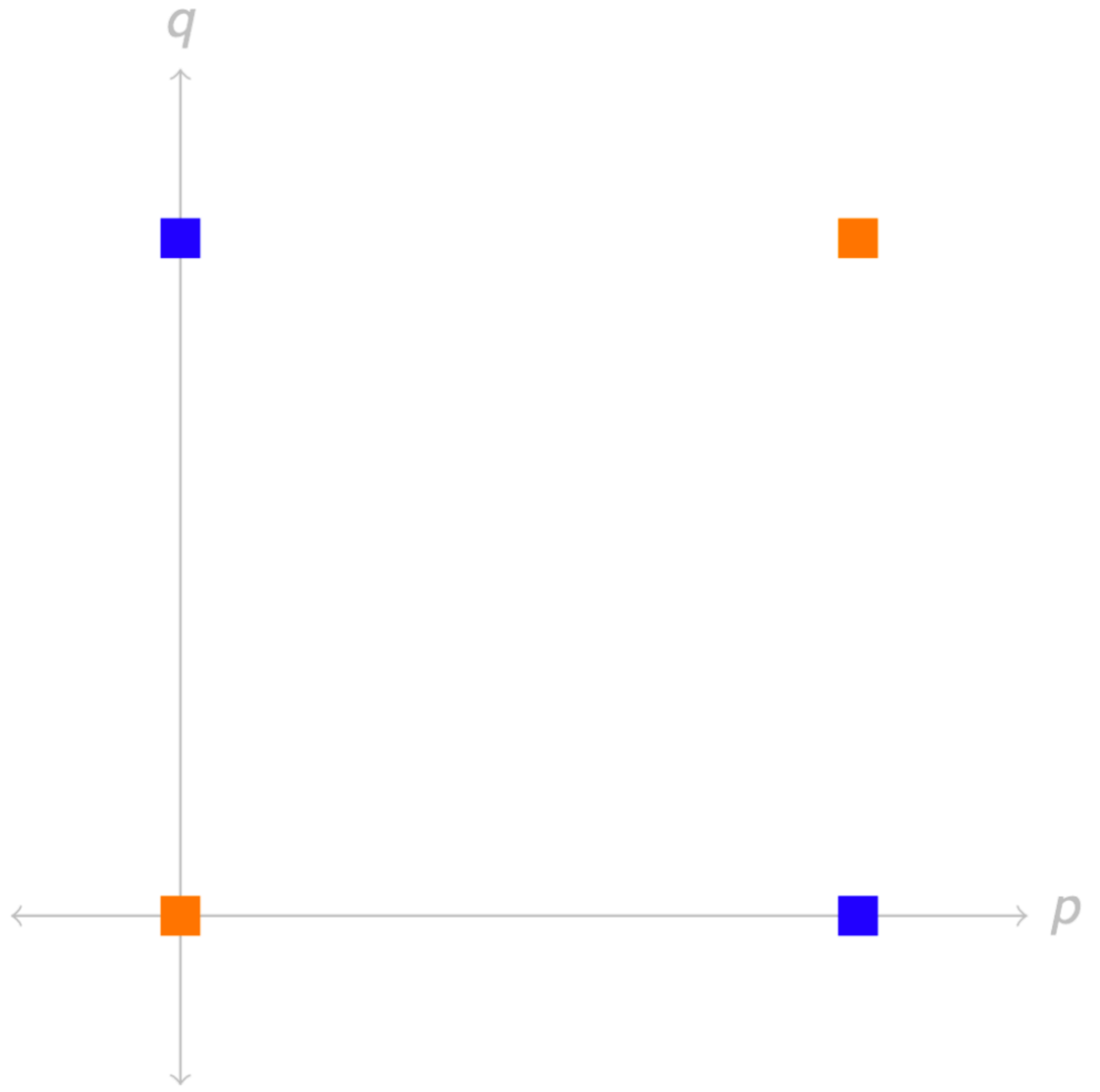
p	q	p XOR q
0	0	0
1	0	1
0	1	1
1	1	0



XOR Function

- The XOR function is True when **either** p or q are True **but not both**
- How would a perceptron model these Truth conditions?

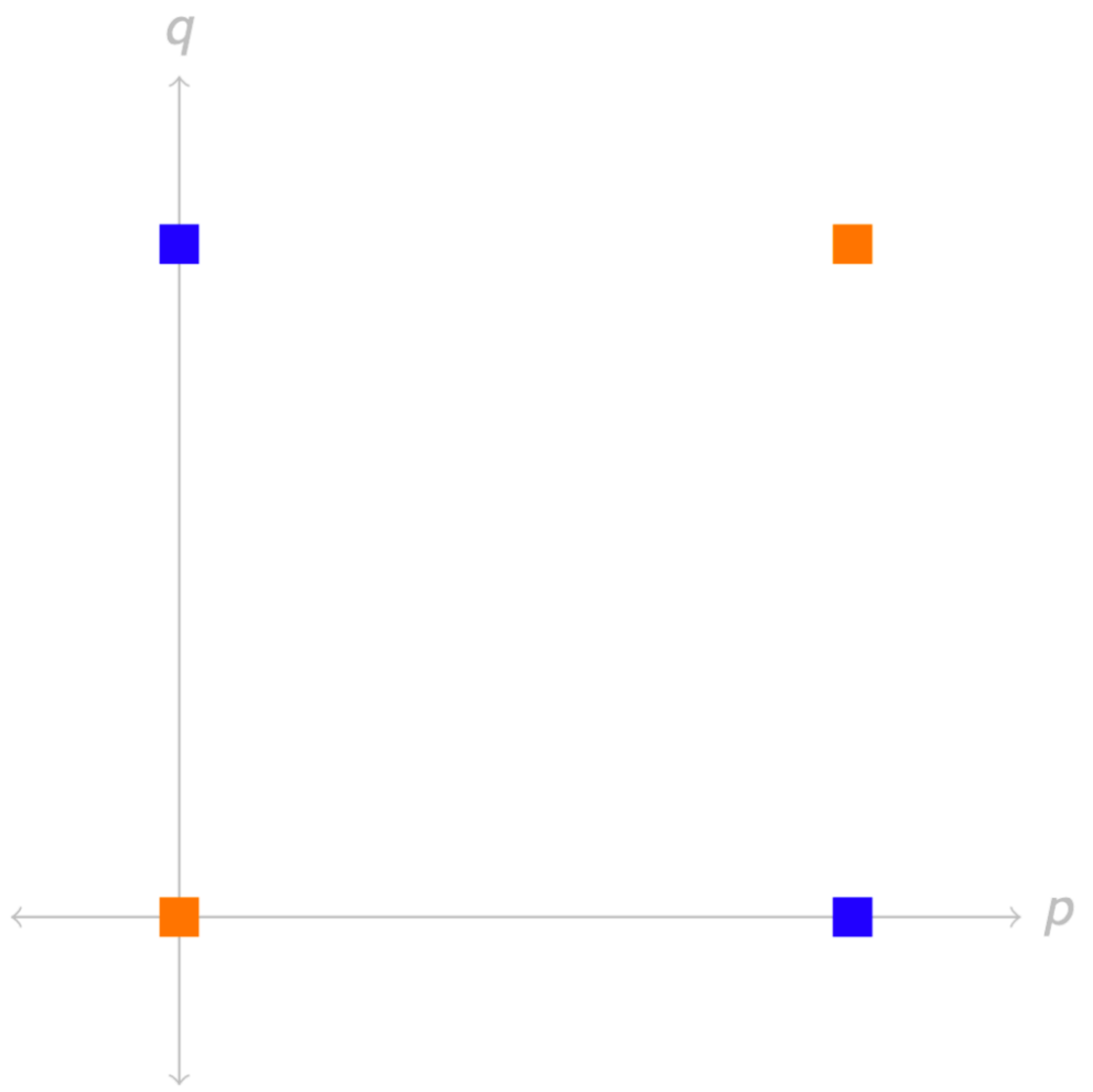
p	q	p XOR q
0	0	0
1	0	1
0	1	1
1	1	0



XOR Function

- The XOR function is True when **either** p or q are True **but not both**
- How would a perceptron model these Truth conditions?
  - **It can't!** The XOR truth conditions are **NOT linearly separable**

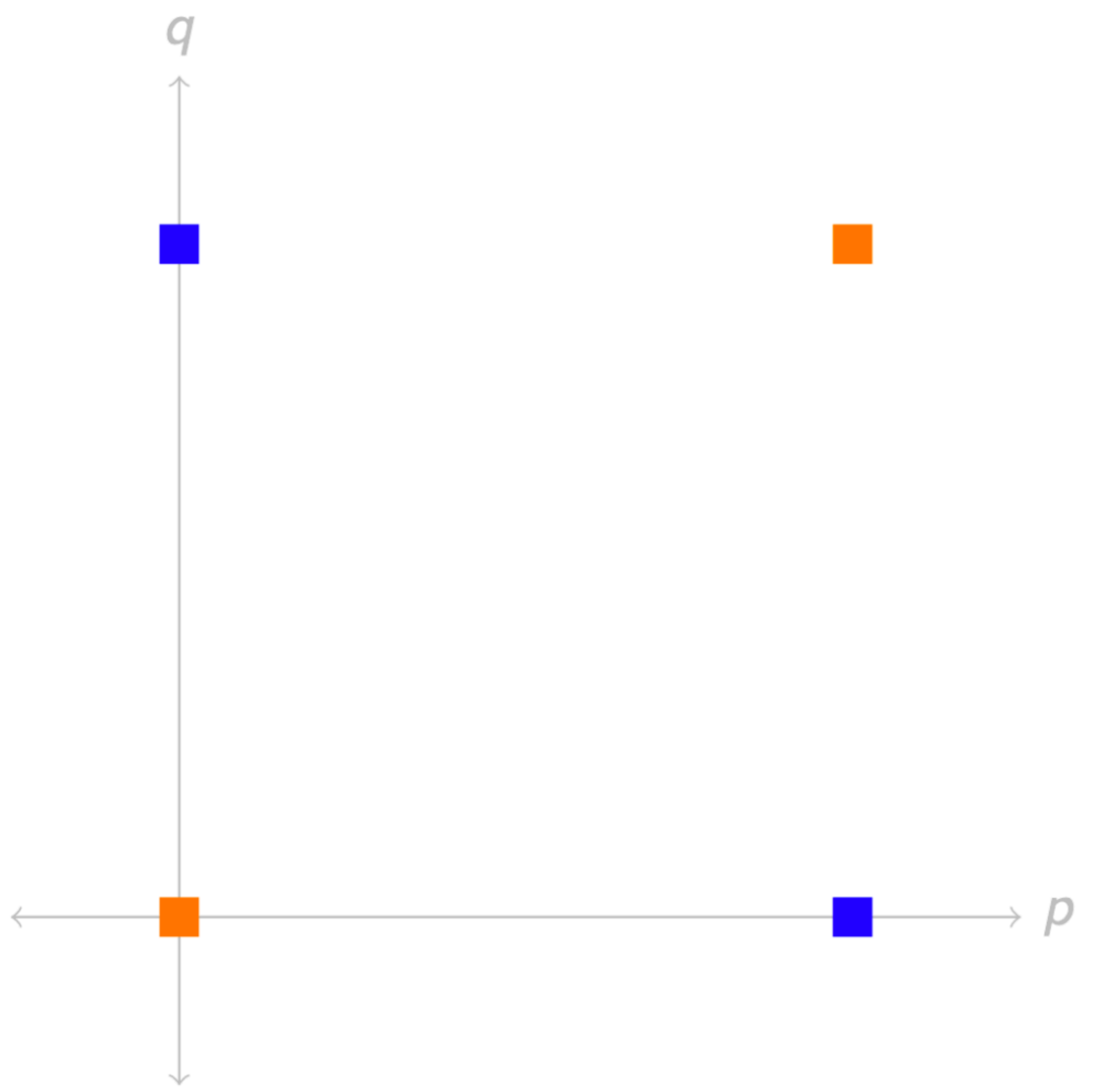
p	q	p XOR q
0	0	0
1	0	1
0	1	1
1	1	0



XOR Function

- The XOR function is True when **either** p or q are True **but not both**
- How would a perceptron model these Truth conditions?
  - **It can't!** The XOR truth conditions are **NOT linearly separable**
  - How do we model functions like this? More in a later lecture

p	q	p XOR q
0	0	0
1	0	1
0	1	1
1	1	0



Last thoughts on Perceptrons

Last thoughts on Perceptrons

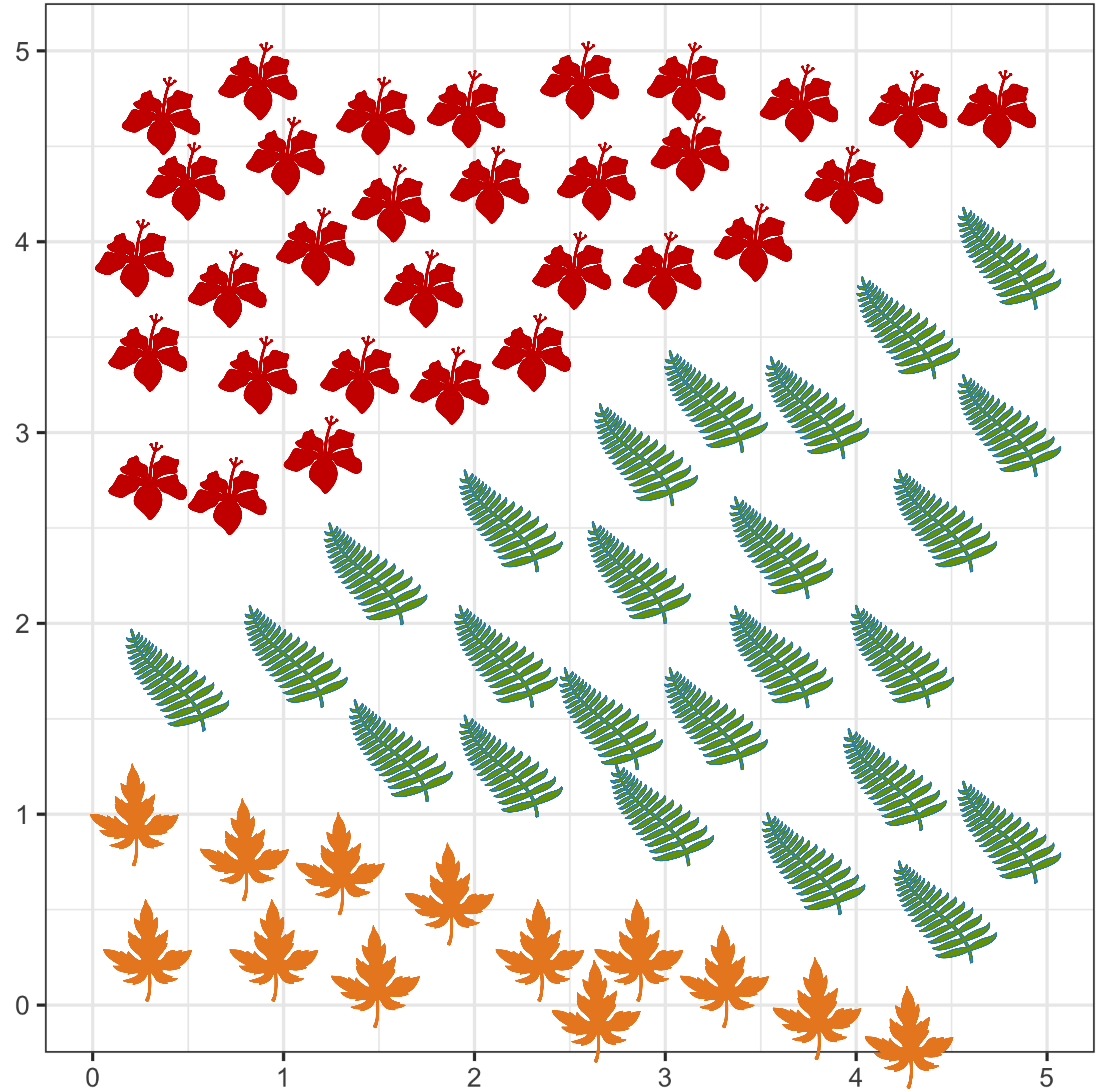
- The Perceptron model goes by **many different names**
  - e.g. Artificial Neuron, **Logistic Regression**, Maximum Entropy (**MaxEnt**)
  - These boil down to the **same model**! (with some difference in training algorithm)

Last thoughts on Perceptrons

- The Perceptron model goes by **many different names**
  - e.g. Artificial Neuron, **Logistic Regression**, Maximum Entropy (**MaxEnt**)
  - These boil down to the **same model**! (with some difference in training algorithm)
- Important points:
  - A Perceptron encodes a **linear separation** in n-dimensional space
  - Like a human neuron, the Perceptron "fires" when the **weighted inputs exceed some threshold**
  - The **weights** need to be **learned from some algorithm** (more next time!)

Practice Example

How do we separate **both** the flowers and the maples from the ferns?



Quiz next time!

- Covering pre-requisites from first-semester Calculus
  - Applying rules to **take the derivative of a function**
    - I will give you a **cheat-sheet** with the rules
  - Extra focus on **how to apply the Chain Rule**
  - Understanding **critical points** of a function/derivative ($f'(x) = 0$)
  - Be able to **sketch a tangent line** at a point on a curve
  - Understanding of **what a function derivative is**
- **No need** for trigonometric functions (cos, sin, tan) or limits right now

Derivatives Cheat-sheet

$$\frac{d}{dx} x^n + c = nx^{n-1}$$

$$\frac{d}{dx} f(x) \pm g(x) = f'(x) \pm g'(x)$$

$$\frac{d}{dx} c \cdot f(x) = c \cdot f'(x)$$

$$\frac{d}{dx} f(x)g(x) = f'(x)g(x) + f(x)g'(x)$$

$$\frac{d}{dx} e^x = e^x$$

$$\frac{d}{dx} \frac{f(x)}{g(x)} = \frac{f'(x)g(x) - f(x)g'(x)}{g(x)^2}$$

$$\frac{d}{dx} ln(x) = \frac{1}{x}$$

Chain Rule: $\frac{d}{dx} f(g(x)) = f'(g(x)) \cdot g'(x)$