Good Morning!

Data is worthless

Data is worthless

Analytics are worth pennies

Data is worthless

Analytics are worth pennies

Decisions are worth dollars

DATA SCIENCE PROFITABILITY PATH

Data is worthless

Analytics are worth pennies

Decisions are worth dollars

Data Science and Machine Learning for Python Folks Without (or With!) a Ph.D.

Douglas Starnes

DataTune 2024

Obligatory Narcissism Slide

Obligatory Narcissism Slide (who is this guy?)

- Hi! I’m Douglas!
- From the Memphis, TN area
  - (working on Nashville)
- Professional explainer
  - Author, speaker & content creator
- User groups
  - Memphis Python & Memphis Azure
- Conferences
  - Scenic City Summit & TDevConf
- 4x Microsoft Most Valuable Professional

Agenda

- Onboarding, prerequisites & tooling
- Python primer
- NumPy
- pandas
- Matplotlib
- scikit-learn
- TensorFlow / Keras

Agenda

- Onboarding, prerequisites & tooling
- Python primer

20%

- NumPy
- pandas
- Matplotlib

50%

- scikit-learn
- TensorFlow / Keras

30%

Expectations

By the end of the day you will

Expectations

By the end of the day you will

- Know the fundamentals of Python and be able to write simple programs

Expectations

By the end of the day you will

- Know the fundamentals of Python and be able to write simple programs
- Be able to use Jupyter notebook, one of the most popular data science tools

Expectations

By the end of the day you will

- Know the fundamentals of Python and be able to write simple programs
- Be able to use Jupyter notebook, one of the most popular data science tools
- Perform basic data analysis using pandas

Expectations

By the end of the day you will

- Know the fundamentals of Python and be able to write simple programs
- Be able to use Jupyter notebook, one of the most popular data science tools
- Perform basic data analysis using pandas
- Know how to create simple visualizations using Matplotlib

Expectations

By the end of the day you will

- Know the fundamentals of Python and be able to write simple programs
- Be able to use Jupyter notebook, one of the most popular data science tools
- Perform basic data analysis using pandas
- Know how to create simple visualizations using Matplotlib
- Understand the concepts of machine learning

Expectations

By the end of the day you will

- Know the fundamentals of Python and be able to write simple programs
- Be able to use Jupyter notebook, one of the most popular data science tools
- Perform basic data analysis using pandas
- Know how to create simple visualizations using Matplotlib
- Understand the concepts of machine learning
- Have a foundation to explore these ideas on your own

Prerequisites

A GitHub account

Prerequisites

A GitHub account

- Personal or organization

Prerequisites

A GitHub account

- Personal or organization
- Free or paid

Prerequisites

A GitHub account

- Personal or organization
- Free or paid
- GitHub Codespaces

Prerequisites

A GitHub account

- Personal or organization
- Free or paid
- GitHub Codespaces
  - Visual Studio Code in the browser

Prerequisites

A GitHub account
- Personal or organization
- Free or paid
- GitHub Codespaces
  - Visual Studio Code in the browser
  - Small Linux VM for compute

Prerequisites

A GitHub account

- Personal or organization
- Free or paid
- GitHub Codespaces
  - Visual Studio Code in the browser
  - Small Linux VM for compute
  - Up to 60 hours free per month

Prerequisites

A GitHub account

- Personal or organization
- Free or paid
- GitHub Codespaces
  - Visual Studio Code in the browser
  - Small Linux VM for compute
  - Up to 60 hours free per month

A Google account

Prerequisites

A GitHub account

- Personal or organization
- Free or paid
- GitHub Codespaces
  - Visual Studio Code in the browser
  - Small Linux VM for compute
  - Up to 60 hours free per month

A Google account

- Personal or organization

Prerequisites

A GitHub account
- Personal or organization
- Free or paid
- GitHub Codespaces
    - Visual Studio Code in the browser
    - Small Linux VM for compute
    - Up to 60 hours free per month

A Google account
- Personal or organization
- Free or paid

Prerequisites

A GitHub account

- Personal or organization
- Free or paid
- GitHub Codespaces
  - Visual Studio Code in the browser
  - Small Linux VM for compute
  - Up to 60 hours free per month

A Google account

- Personal or organization
- Free or paid
- Google Colab

Prerequisites

A GitHub account

- Personal or organization
- Free or paid
- GitHub Codespaces
  - Visual Studio Code in the browser
  - Small Linux VM for compute
  - Up to 60 hours free per month

A Google account

- Personal or organization
- Free or paid
- Google Colab
  - Hosting Jupyter notebooks

Prerequisites

A GitHub account
- Personal or organization
- Free or paid
- GitHub Codespaces
  - Visual Studio Code in the browser
  - Small Linux VM for compute
  - Up to 60 hours free per month

A Google account
- Personal or organization
- Free or paid
- Google Colab
  - Hosting Jupyter notebooks
  - Free GPU access

I apologize in advance for any

I apologize in advance for any

MATH

Python

According to the Octoverse (GitHub), the second most popular programming language in the world second only to JavaScript.

This means it is the top real programming language.

Variables

Variables

name = value

Variables

`name = value`

no keywords
(var, let, etc.)

Variables

name = value

no type
specifiers

Variables

name = value

no semicolons

Variables

```
name = value
foo = 42
```

Variables

name = value

```
foo = 42
```

declare a variable `foo` and initialize it with the value 42

Variables

name = value

```
foo = 42
```

42 is of type integer (`int`)
so `foo` is of type `int`

Variables

```
name = value
foo = 42
foo = “forty two”
```

Variables

`name = value`

```
foo = 42
foo = “forty two”
```

“forty two” is of type string (`str`) so `foo` is of type `str`

Variables

```
name = value
foo = 42
foo = “forty two”
```

Python is **dynamically typed** thus the type of the variable changes with the type of the value

Variables

name = value

```
foo = 42
```

```
foo = “forty two”
```

```
uninitialized
uninitialized = “initial value”
```

Variables

```
name = value

foo = 42

foo = “forty two”

uninitialized
uninitialized = “initial value”
```

Don’t do this!

Variables

```
name = value
foo = 42
foo = “forty two”
uninitialized
uninitialized = “initial value”
```

You must initialize a variable when it is declared

Types

Types

Numeric

- integer (`int`)
- float (`float`)
- complex (`complex`)

Types

Numeric
- integer (`int`)
- float (`float`)
- complex (`complex`)

string (`str`)

Types

Numeric
- integer (`int`)
- float (`float`)
- complex (`complex`)

string (`str`)

strings can be double or single quoted

Types

Numeric
- integer (`int`)
- float (`float`)
- complex (`complex`)

string (`str`)

there is no character type

Types

Numeric
- integer (`int`)
- float (`float`)
- complex (`complex`)

string (`str`)

there is no character type
`'A'` and `"A"` are equivalent strings

Commercial break: f-strings

Commercial break: f-strings

There are several method for formatting a string in Python. In this workshop I'll use the "f-string".

Commercial break: f-strings

There are several method for formatting a string in Python. In this workshop I'll use the "f-string".

The f-string is preceded, outside the quotes, with the letter 'f'.

Commercial break: f-strings

There are several method for formatting a string in Python. In this workshop I’ll use the “f-string”.

The f-string is preceded, outside the quotes, with the letter ‘f’.

```
greeting = “Hi there!”
my_string = f””
```

Commercial break: f-strings

There are several method for formatting a string in Python. In this workshop I'll use the "f-string".

The f-string is preceded, outside the quotes, with the letter 'f'.

Place Python expressions inside curly braces and they will be evaluated and inserted

```
greeting = “Hi there!”
my_string = f””
```

Commercial break: f-strings

There are several method for formatting a string in Python. In this workshop I'll use the "f-string".

The f-string is preceded, outside the quotes, with the letter 'f'.

```
greeting = "Hi there!"
my_string = f""
```

Place Python expressions inside curly braces and they will be evaluated and inserted

```
my_string =
  f"{greeting} 1 + 1 = {1 + 1}"
print(my_string) # "Hi there! 1 + 1 = 2"
```

Commercial break: f-strings

There are several method for formatting a string in Python. In this workshop I'll use the "f-string".

The f-string is preceded, outside the quotes, with the letter 'f'.

```
greeting = “Hi there!”
my_string = f””
```

Place Python expressions inside curly braces and they will be evaluated and inserted

```
my_string =
  f”{greeting} 1 + 1 = {1 + 1}”
print(my_string) # “Hi there! 1 + 1 = 2”
```

btw ... this is how you create a comment in Python

Types

Numeric
- integer (`int`)
- float (`float`)
- complex (`complex`)

string (`str`)

Boolean (`bool`)
- `True`
- `False`

Types

Numeric
- integer (`int`)
- float (`float`)
- complex (`complex`)

string (`str`)

Boolean (`bool`)
- `True`
- `False`

null value (`None`)

Collection Types

Collection Types

List (`list`)

Collection Types

List (`list`)

- ordered, linear collection of Python values

Collection Types

List (`list`)

- ordered, linear collection of Python values
- values are separated by commas

Collection Types

List (`list`)

- ordered, linear collection of Python values
- values are separated by commas
- the list is surrounded by square brackets

Collection Types

List (`list`)

- ordered, linear collection of Python values
- values are separated by commas
- the list is surrounded by square brackets

```
my_list = [“foo”, 42, True, None]
```

Collection Types

List (`list`)

- ordered, linear collection of Python values
- values are separated by commas
- the list is surrounded by square brackets

Tuple (`tuple`)

```
my_list = [“foo”, 42, True, None]
```

Collection Types

List (`list`)
- ordered, linear collection of Python values
- values are separated by commas
- the list is surrounded by square brackets

Tuple (`tuple`)
- ordered, linear collection of Python values

```
my_list = [“foo”, 42, True, None]
```

Collection Types

List (`list`)
- ordered, linear collection of Python values
- values are separated by commas
- the list is surrounded by square brackets

Tuple (`tuple`)
- ordered, linear collection of Python values
- values are separated by commas

```
my_list = [“foo”, 42, True, None]
```

Collection Types

List (`list`)
- ordered, linear collection of Python values
- values are separated by commas
- the list is surrounded by square brackets

Tuple (`tuple`)
- ordered, linear collection of Python values
- values are separated by commas
- the tuple is surrounded by parentheses

```
my_list = [“foo”, 42, True, None]
```

Collection Types

List (`list`)
- ordered, linear collection of Python values
- values are separated by commas
- the list is surrounded by square brackets

Tuple (`tuple`)
- ordered, linear collection of Python values
- values are separated by commas
- the tuple is surrounded by parentheses

```
my_list = [“foo”, 42, True, None]
```

```
my_tuple = (“foo”, 42, True, None)
```

Collection Types

List (`list`)
- ordered, linear collection of Python values
- values are separated by commas
- the list is surrounded by square brackets

```
my_list = [“foo”, 42, True, None]
```

Tuple (`tuple`)
- ordered, linear collection of Python values
- values are separated by commas
- the tuple is surrounded by parentheses

```
my_tuple = (“foo”, 42, True, None)
```

Dictionary (`dict`)

Collection Types

List (`list`)
- ordered, linear collection of Python values
- values are separated by commas
- the list is surrounded by square brackets

```
my_list = [“foo”, 42, True, None]
```

Tuple (`tuple`)
- ordered, linear collection of Python values
- values are separated by commas
- the tuple is surrounded by parentheses

```
my_tuple = (“foo”, 42, True, None)
```

Dictionary (`dict`)
- collection of key/value pairs

Collection Types

List (`list`)
- ordered, linear collection of Python values
- values are separated by commas
- the list is surrounded by square brackets

Tuple (`tuple`)
- ordered, linear collection of Python values
- values are separated by commas
- the tuple is surrounded by parentheses

Dictionary (`dict`)
- collection of key/value pairs
- keys and values are separated by colons

```
my_list = [“foo”, 42, True, None]
```

```
my_tuple = (“foo”, 42, True, None)
```

Collection Types

List (`list`)
- ordered, linear collection of Python values
- values are separated by commas
- the list is surrounded by square brackets

```
my_list = [“foo”, 42, True, None]
```

Tuple (`tuple`)
- ordered, linear collection of Python values
- values are separated by commas
- the tuple is surrounded by parentheses

```
my_tuple = (“foo”, 42, True, None)
```

Dictionary (`dict`)
- collection of key/value pairs
- keys and values are separated by colons
- pairs are separated by commas

Collection Types

List (`list`)
- ordered, linear collection of Python values
- values are separated by commas
- the list is surrounded by square brackets

```
my_list = [“foo”, 42, True, None]
```

Tuple (`tuple`)
- ordered, linear collection of Python values
- values are separated by commas
- the tuple is surrounded by parentheses

```
my_tuple = (“foo”, 42, True, None)
```

Dictionary (`dict`)
- collection of key/value pairs
- keys and values are separated by colons
- pairs are separated by commas
- the whole thing is surrounded by curly braces

Collection Types

List (`list`)
- ordered, linear collection of Python values
- values are separated by commas
- the list is surrounded by square brackets

```
my_list = [“foo”, 42, True, None]
```

Tuple (`tuple`)
- ordered, linear collection of Python values
- values are separated by commas
- the tuple is surrounded by parentheses

```
my_tuple = (“foo”, 42, True, None)
```

Dictionary (`dict`)
- collection of key/value pairs
- keys and values are separated by colons
- pairs are separated by commas
- the whole thing is surrounded by curly braces

```
my_dict = {
    “string”: “foo”,
    “int”: 42,
    “boolean”: True,
    “null”: None
}
```

List vs. Tuple

List vs. Tuple

Zero indexed (first element is at index 0)

List vs. Tuple

```
my_list = [“one”, “two”, “three”, “four”]
```

Zero indexed (first element is at index 0)

List vs. Tuple

Zero indexed (first element is at index 0)

```
my_list = [“one”, “two”, “three”, “four”]

my_list[0] == “one”
```

List vs. Tuple

```
my_list = [“one”, “two”, “three”, “four”]
```

Zero indexed (first element is at index 0)

```
my_list[0] == “one”
```

Negative indexing starts at the last element and counts backwards

List vs. Tuple

```
my_list = [“one”, “two”, “three”, “four”]
```

Zero indexed (first element is at index 0)

```
my_list[0] == “one”
```

Negative indexing starts at the last element and counts backwards

```
my_list[-1] == “four”
```

List vs. Tuple

Zero indexed (first element is at index 0)

Negative indexing starts at the last element and counts backwards

Slicing

```
my_list = [“one”, “two”, “three”, “four”]
my_list[0] == “one”
my_list[-1] == “four”
```

List vs. Tuple

```
my_list = [“one”, “two”, “three”, “four”]
```

Zero indexed (first element is at index 0)

```
my_list[0] == “one”
```

Negative indexing starts at the last element and counts backwards

```
my_list[-1] == “four”
```

Slicing

- *[lower:upper]*

List vs. Tuple

Zero indexed (first element is at index 0)

Negative indexing starts at the last element and counts backwards

Slicing
- *[lower:upper]*
- Lower bound is inclusive, upper bound is exclusive

```
my_list = [“one”, “two”, “three”, “four”]
my_list[0] == “one”
my_list[-1] == “four”
```

List vs. Tuple

Zero indexed (first element is at index 0)

Negative indexing starts at the last element and counts backwards

Slicing
- *[lower:upper]*
- Lower bound is inclusive, upper bound is exclusive

```
my_list = [“one”, “two”, “three”, “four”]
my_list[0] == “one”
my_list[-1] == “four”
my_list[1:3] == [“two”, “three”]
```

List vs. Tuple

```
my_list = [“one”, “two”, “three”, “four”]
```

Zero indexed (first element is at index 0)

```
my_list[0] == “one”
```

Negative indexing starts at the last element and counts backwards

```
my_list[-1] == “four”
```

Slicing
- *[lower:upper]*
- Lower bound is inclusive, upper bound is exclusive
- Omitting lower bound starts at the first element

```
my_list[1:3] == [“two”, “three”]
```

List vs. Tuple

```
my_list = [“one”, “two”, “three”, “four”]
```

Zero indexed (first element is at index 0)

```
my_list[0] == “one”
```

Negative indexing starts at the last element and counts backwards

```
my_list[-1] == “four”
```

Slicing

- *`[lower:upper]`*
- Lower bound is inclusive, upper bound is exclusive
- Omitting lower bound starts at the first element

```
my_list[1:3] == [“two”, “three”]

my_list[:2] == [“one”, “two”]
```

List vs. Tuple

Zero indexed (first element is at index 0)

Negative indexing starts at the last element and counts backwards

Slicing
- *[lower:upper]*
- Lower bound is inclusive, upper bound is exclusive
- Omitting lower bound starts at the first element
- Omitting upper bound stops at the last element

```
my_list = [“one”, “two”, “three”, “four”]

my_list[0] == “one”

my_list[-1] == “four”

my_list[1:3] == [“two”, “three”]

my_list[:2] == [“one”, “two”]
```

List vs. Tuple

```
my_list = [“one”, “two”, “three”, “four”]
```

Zero indexed (first element is at index 0)

```
my_list[0] == “one”
```

Negative indexing starts at the last element and counts backwards

```
my_list[-1] == “four”
```

Slicing
- *[lower:upper]*
- Lower bound is inclusive, upper bound is exclusive
- Omitting lower bound starts at the first element
- Omitting upper bound stops at the last element

```
my_list[1:3] == [“two”, “three”]

my_list[:2] == [“one”, “two”]

my_list[-2:]
```

List vs. Tuple

```
my_list = [“one”, “two”, “three”, “four”]
```

Zero indexed (first element is at index 0)

```
my_list[0] == “one”
```

Negative indexing starts at the last element and counts backwards

```
my_list[-1] == “four”
```

Slicing
- *[lower:upper]*
- Lower bound is inclusive, upper bound is exclusive
- Omitting lower bound starts at the first element
- Omitting upper bound stops at the last element

```
my_list[1:3] == [“two”, “three”]
my_list[:2] == [“one”, “two”]
my_list[-2:] == [“three”, “four”]
```

List vs. Tuple

Zero indexed (first element is at index 0)

Negative indexing starts at the last element and counts backwards

Slicing
- *[lower:upper]*
- Lower bound is inclusive, upper bound is exclusive
- Omitting lower bound starts at the first element
- Omitting upper bound stops at the last element

The `len()` function returns the number of elements

```
my_list = [“one”, “two”, “three”, “four”]
my_list[0] == “one”
my_list[-1] == “four”
my_list[1:3] == [“two”, “three”]
my_list[:2] == [“one”, “two”]
my_list[-2:] == [“three”, “four”]
```

List vs. Tuple

`my_list = [“one”, “two”, “three”, “four”]`

Zero indexed (first element is at index 0)

`my_list[0] == “one”`

Negative indexing starts at the last element and counts backwards

`my_list[-1] == “four”`

Slicing
- *[lower:upper]*
- Lower bound is inclusive, upper bound is exclusive
- Omitting lower bound starts at the first element
- Omitting upper bound stops at the last element

`my_list[1:3] == [“two”, “three”]`

`my_list[:2] == [“one”, “two”]`

`my_list[-2:] == [“three”, “four”]`

The `len()` function returns the number of elements

`len(my_list) == 4`

List vs. Tuple

List vs. Tuple

Assign a new value to an index in the list

List vs. Tuple

Assign a new value to an index in the list

```
my_list[0] == “zero”
```

List vs. Tuple

Assign a new value to an index in the list

`my_list[0] == “zero”`

Add a new value to the end of the list

List vs. Tuple

Assign a new value to an index in the list	`my_list[0] == “zero”`
Add a new value to the end of the list	`my_list.append(“five”)`

List vs. Tuple

Assign a new value to an index in the list

```
my_list[0] == “zero”
```

Add a new value to the end of the list

```
my_list.append(“five”)
```

Add multiple values to the end of the list

List vs. Tuple

Assign a new value to an index in the list	`my_list[0] == “zero”`
Add a new value to the end of the list	`my_list.append(“five”)`
Add multiple values to the end of the list	`my_list.extend([“five”, “six”, “seven”])`

List vs. Tuple

Assign a new value to an index in the list	`my_list[0] == “zero”`
Add a new value to the end of the list	`my_list.append(“five”)`
Add multiple values to the end of the list	`my_list.extend([“five”, “six”, “seven”])`
Remove a value from the list	

List vs. Tuple

Assign a new value to an index in the list	`my_list[0] == “zero”`
Add a new value to the end of the list	`my_list.append(“five”)`
Add multiple values to the end of the list	`my_list.extend([“five”, “six”, “seven”])`
Remove a value from the list	`my_list.remove(“four”)`

List vs. Tuple

Assign a new value to an index in the list	`my_list[0] == “zero”`
Add a new value to the end of the list	`my_list.append(“five”)`
Add multiple values to the end of the list	`my_list.extend([“five”, “six”, “seven”])`
Remove a value from the list	`my_list.remove(“four”)`

List vs. Tuple

Assign a new value to an index in the list	`my_list[0] == “zero”`
Add a new value to the end of the list	`my_list.append(“five”)`
Add multiple values to the end of the list	`my_list.extend([“five”, “six”, “seven”])`
Remove a value from the list	`my_list.remove(“four”)`

You cannot modify tuples, they are fixed-length and immutable.

List vs. Tuple

```
("one", "two", "three")
```

List vs. Tuple

```
a, b, c = (“one”, “two”, “three”)
```

List vs. Tuple

```
a, b, c = (“one”, “two”, “three”)
```

List vs. Tuple

```
a, b, c = (“one”, “two”, “three”)
```

“Unpacking” a tuple

List vs. Tuple

```
("one", "two", "three")
```

List vs. Tuple

```
a, b, _ = (“one”, “two”, “three”)
```

List vs. Tuple

```
a, b, _ = (“one”, “two”, “three”)
```

“Throwaway” variable

List vs. Tuple

```
[]        [42,]        [42]

()        (42,)        (42)
```

List vs. Tuple

`[]`	`[42,]`	`[42]`
Empty list		
`()`	`(42,)`	`(42)`

List vs. Tuple

`[]` `[42,]` `[42]`

Empty list

Empty tuple

`()` `(42,)` `(42)`

List vs. Tuple

List with a single element

```
[]          [42,]          [42]
```

Empty list

Empty tuple

```
()          (42,)          (42)
```

List vs. Tuple

List with a single element

```
[]        [42,]        [42]
```

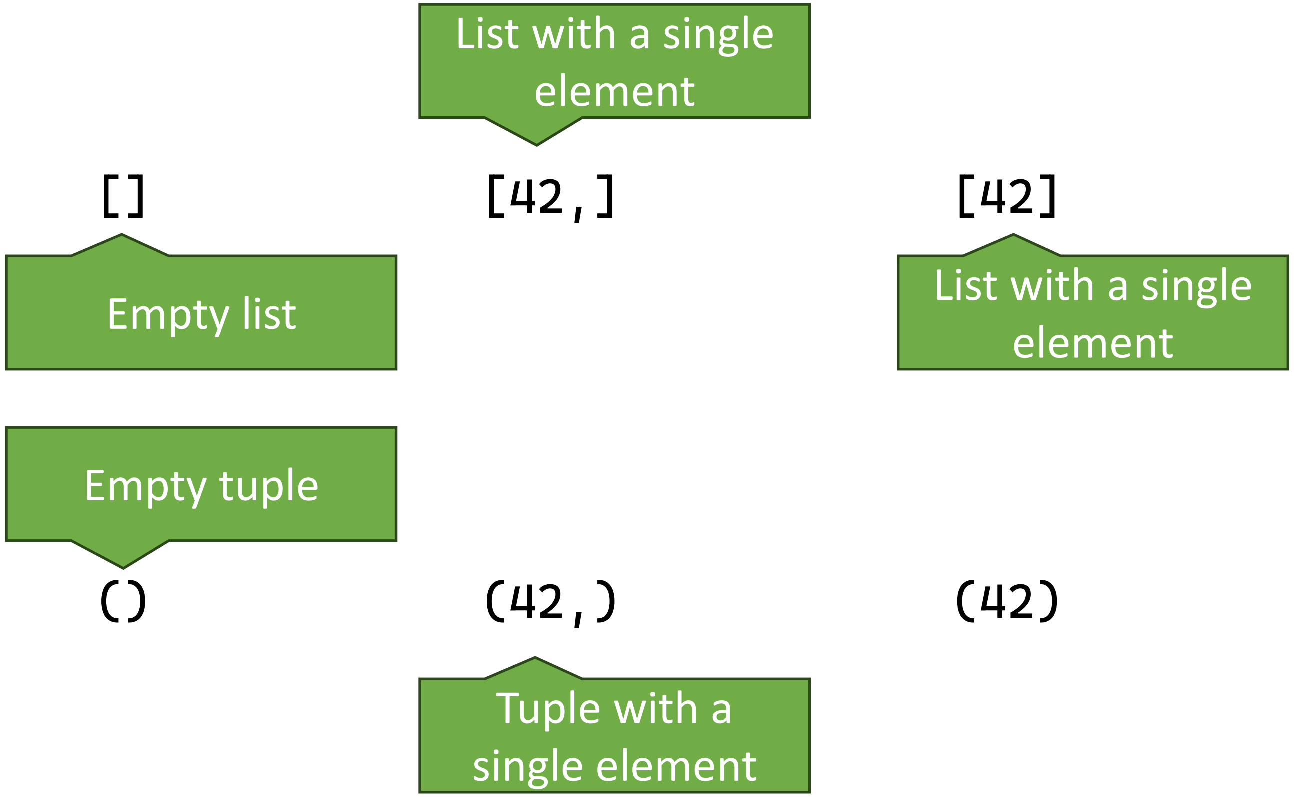
Empty list

Empty tuple

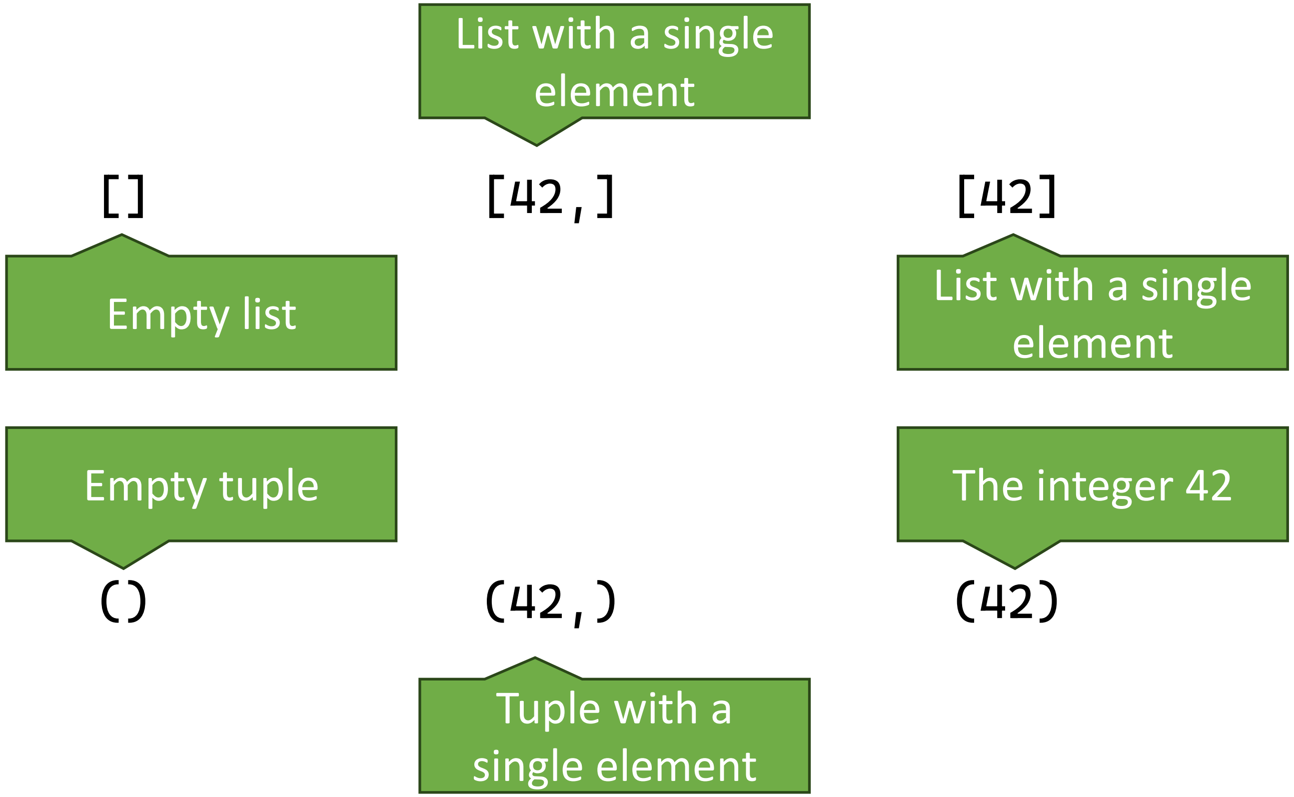
```
()        (42,)        (42)
```

Tuple with a single element

List vs. Tuple



List vs. Tuple



Dictionaries

Dictionaries

```
my_dict = {
    “string”: “foo”,
    “int”: 42,
    “boolean”: True,
    “null”: None
}
```

Dictionaries

Access a value by key in square brackets

```
my_dict = {
    “string”: “foo”,
    “int”: 42,
    “boolean”: True,
    “null”: None
}
```

Dictionaries

Access a value by key in square brackets

```
my_dict[“int”] == 42
```

```
my_dict = {
    “string”: “foo”,
    “int”: 42,
    “boolean”: True,
    “null”: None
}
```

Dictionaries

Access a value by key in square brackets

```
my_dict[“int”] == 42
```

Assign a new value to a key

```
my_dict = {
    “string”: “foo”,
    “int”: 42,
    “boolean”: True,
    “null”: None
}
```

Dictionaries

Access a value by key in square brackets

```
my_dict[“int”] == 42
```

Assign a new value to a key

```
my_dict[“int”] = 100
```

```
my_dict = {
    “string”: “foo”,
    “int”: 42,
    “boolean”: True,
    “null”: None
}
```

Dictionaries

Access a value by key in square brackets

```
my_dict[“int”] == 42
```

Assign a new value to a key

```
my_dict[“int”] = 100
```

Add a new key dynamically

```
my_dict = {
    “string”: “foo”,
    “int”: 42,
    “boolean”: True,
    “null”: None
}
```

Dictionaries

Access a value by key in square brackets

```
my_dict[“int”] == 42
```

Assign a new value to a key

```
my_dict[“int”] = 100
```

Add a new key dynamically

```
my_dict[“list”] =[“a”, “new”, “list”]
```

```
my_dict = {
    “string”: “foo”,
    “int”: 42,
    “boolean”: True,
    “null”: None
}
```

Loops

Loops

Python does not implement the C-style for-next loop

Loops

Python does not implement the C-style for-next loop

Instead of indexing through collections, it iterates over the elements in the collection

Loops

Python does not implement the C-style for-next loop

Instead of indexing through collections, it iterates over the elements in the collection

```
for element in my_list:
```

Loops

Python does not implement the C-style for-next loop

Instead of indexing through collections, it iterates over the elements in the collection

The colon indicates the body of the for loop begins on the next line. The body **must** be indented

```
for element in my_list:
```

Loops

Python does not implement the C-style for-next loop

Instead of indexing through collections, it iterates over the elements in the collection

```
for element in my_list:
```

The colon indicates the body of the for loop begins on the next line. The body **must** be indented

```
for element in my_list:
  print(element)
```

Commercial break for indentation & PEP-8

Commercial break for indentation & PEP-8

In Python, indentation denotes scope

Commercial break for indentation & PEP-8

In Python, indentation denotes scope

Indentation must be consistent

Commercial break for indentation & PEP-8

In Python, indentation denotes scope

Indentation must be consistent
- type (tabs or spaces)

Commercial break for indentation & PEP-8

In Python, indentation denotes scope

Indentation must be consistent
- type (tabs or spaces)
- width

Commercial break for indentation & PEP-8

In Python, indentation denotes scope

Indentation must be consistent
- type (tabs or spaces)
- width

If you mix tabs and spaces, or have multiple widths, the code will crash

Commercial break for indentation & PEP-8

In Python, indentation denotes scope

Indentation must be consistent
- type (tabs or spaces)
- width

If you mix tabs and spaces, or have multiple widths, the code will crash

This is supposed to help make code readable

Commercial break for indentation & PEP-8

In Python, indentation denotes scope

Indentation must be consistent
- type (tabs or spaces)
- width

If you mix tabs and spaces, or have multiple widths, the code will crash

This is supposed to help make code readable

PEP-8, a style recommendation, suggests 4 spaces

Commercial break for indentation & PEP-8

In Python, indentation denotes scope

Indentation must be consistent
- type (tabs or spaces)
- width

If you mix tabs and spaces, or have multiple widths, the code will crash

This is supposed to help make code readable

PEP-8, a style recommendation, suggests 4 spaces

In the slides, I'll be using 2 spaces to conserve screen real estate, but for readability the code examples will use 4 spaces

Loops

Python does not implement the C-style for-next loop

Instead of indexing through collections, it iterates over the elements in the collection

```
for element in my_list:
```

The colon indicates the body of the for loop begins on the next line. The body **must** be indented

```
for element in my_list:
  print(element)
```

Loops

Python does not implement the C-style for-next loop

Instead of indexing through collections, it iterates over the elements in the collection

The colon indicates the body of the for loop begins on the next line. The body **must** be indented

```
for element in my_list:
```

```
for element in my_list:
  print(element)
```

No parentheses

Loops

Python does not implement the C-style for-next loop

Instead of indexing through collections, it iterates over the elements in the collection

```
for element in my_list:
```

The colon indicates the body of the for loop begins on the next line. The body **must** be indented

```
for element in my_list:
  print(element)
```

Looping a dictionary will iterate over the keys

Loops

Python does not implement the C-style for-next loop

Instead of indexing through collections, it iterates over the elements in the collection

```
for element in my_list:
```

The colon indicates the body of the for loop begins on the next line. The body **must** be indented

```
for element in my_list:
  print(element)
```

Looping a dictionary will iterate over the keys

```
for key in my_dict:
  print(my_dict[key])
```

Loops

Python does not implement the C-style for-next loop

Instead of indexing through collections, it iterates over the elements in the collection

```
for element in my_list:
```

The colon indicates the body of the for loop begins on the next line. The body **must** be indented

```
for element in my_list:
  print(element)
```

Looping a dictionary will iterate over the keys

```
for key in my_dict:
  print(my_dict[key])
```

If you need the indices of the elements, use the enumerate function. It will return a tuple with the index and element that can be unpacked.

Loops

Python does not implement the C-style for-next loop

Instead of indexing through collections, it iterates over the elements in the collection

```
for element in my_list:
```

The colon indicates the body of the for loop begins on the next line. The body **must** be indented

```
for element in my_list:
  print(element)
```

Looping over a dictionary will iterate the keys

```
for key in my_dict:
  print(my_dict[key])
```

If you need the indices of the elements, use the enumerate function. It will return a tuple with the index and element that can be unpacked.

```
for idx, el in enumerate(my_list):
  print(f"{idx + 1} – {el}")
```

Loops

Python does not implement the C-style for-next loop

Instead of indexing through collections, it iterates over the elements in the collection

```
for element in my_list:
```

The colon indicates the body of the for loop begins on the next line. The body **must** be indented

```
for element in my_list:
  print(element)
```

Looping over a dictionary will iterate the keys

```
for key in my_dict:
  print(my_dict[key])
```

If you need the indices of the elements, use the enumerate function. It will return a tuple with the index and element that can be unpacked.

```
for idx, el in enumerate(my_list):
  print(f"{idx + 1} – {el}")
```

The items method of the dictionary will for each key in the dictionary return the key and value in a tuple that can be unpacked.

Loops

Python does not implement the C-style for-next loop

Instead of indexing through collections, it iterates over the elements in the collection

```
for element in my_list:
```

The colon indicates the body of the for loop begins on the next line. The body **must** be indented

```
for element in my_list:
  print(element)
```

Looping over a dictionary will iterate the keys

```
for key in my_dict:
  print(my_dict[key])
```

If you need the indices of the elements, use the enumerate function. It will return a tuple with the index and element that can be unpacked.

```
for idx, el in enumerate(my_list):
  print(f"{idx + 1} – {el}")
```

The items method of the dictionary will for each key in the dictionary return the key and value in a tuple that can be unpacked.

```
for key, value in my_dict.items():
  print(f"{key} – {value}")
```

Conditionals

Conditionals

Python implements the if keyword for conditionals

Conditionals

Python implements the if keyword for conditionals

```
if coin_flip == “heads”:
```

Conditionals

Python implements the if keyword for conditionals

The body to be executed if the conditional is True, must be indented like the blocks for loops

```
if coin_flip == “heads”:
```

Conditionals

Python implements the if keyword for conditionals

The body to be executed if the conditional is True, must be indented like the blocks for loops

```
if coin_flip == “heads”:

if coin_flip == 0:
  print(“The flip is heads”)
```

Conditionals

Python implements the if keyword for conditionals

The body to be executed if the conditional is True, must be indented like the blocks for loops

The branch to be executed if the conditional is False, can be added with an else clause

```
if coin_flip == “heads”:

if coin_flip == 0:
  print(“The flip is heads”)
```

Conditionals

Python implements the if keyword for conditionals

```
if coin_flip == “heads”:
```

The body to be executed if the conditional is True, must be indented like the blocks for loops

```
if coin_flip == 0:
  print(“The flip is heads”)
```

The branch to be executed if the conditional is False, can be added with an else clause

```
if coin_flip == 0:
  print(“The flip is heads”)
else:
  print(“The flip is tails”)
```

Conditionals

Python implements the if keyword for conditionals

```
if coin_flip == “heads”:
```

The body to be executed if the conditional is True, must be indented like the blocks for loops

```
if coin_flip == 0:
  print(“The flip is heads”)
```

The branch to be executed if the conditional is False, can be added with an else clause

```
if coin_flip == 0:
  print(“The flip is heads”)
else:
  print(“The flip is tails”)
```

Multiple branches can be added with elif (short for else if)

Conditionals

Python implements the if keyword for conditionals

```
if coin_flip == “heads”:
```

The body to be executed if the conditional is True, must be indented like the blocks for loops

```
if coin_flip == 0:
  print(“The flip is heads”)
```

The branch to be executed if the conditional is False, can be added with an else clause

```
if coin_flip == 0:
  print(“The flip is heads”)
else:
  print(“The flip is tails”)
```

Multiple branches can be added with elif (short for else if)

```
if coin_flip == 0:
  print(“The flip is heads”)
elif coin_flip == 1:
  print(“The flip is tails”)
else:
  print(“What kind of coin are you using?”)
```

Conditionals

Python implements the if keyword for conditionals

```
if coin_flip == “heads”:
```

The body to be executed if the conditional is True, must be indented like the blocks for loops

```
if coin_flip == 0:
  print(“The flip is heads”)
```

The branch to be executed if the conditional is False, can be added with an else clause

```
if coin_flip == 0:
  print(“The flip is heads”)
else:
  print(“The flip is tails”)
```

Multiple branches can be added with elif (short for else if)

```
if coin_flip == 0:
  print(“The flip is heads”)
elif coin_flip == 1:
  print(“The flip is tails”)
else:
  print(“What kind of coin are you using?”)
```

There is no switch statement

Let’s Build an App!

Let’s Build an App!

Our first project will be a (simple) application to manage investments in a cryptocurrency portfolio

Let’s Build an App!

Our first project will be a (simple) application to manage investments in a cryptocurrency portfolio

What kinds of data do we need to store?

Let’s Build an App!

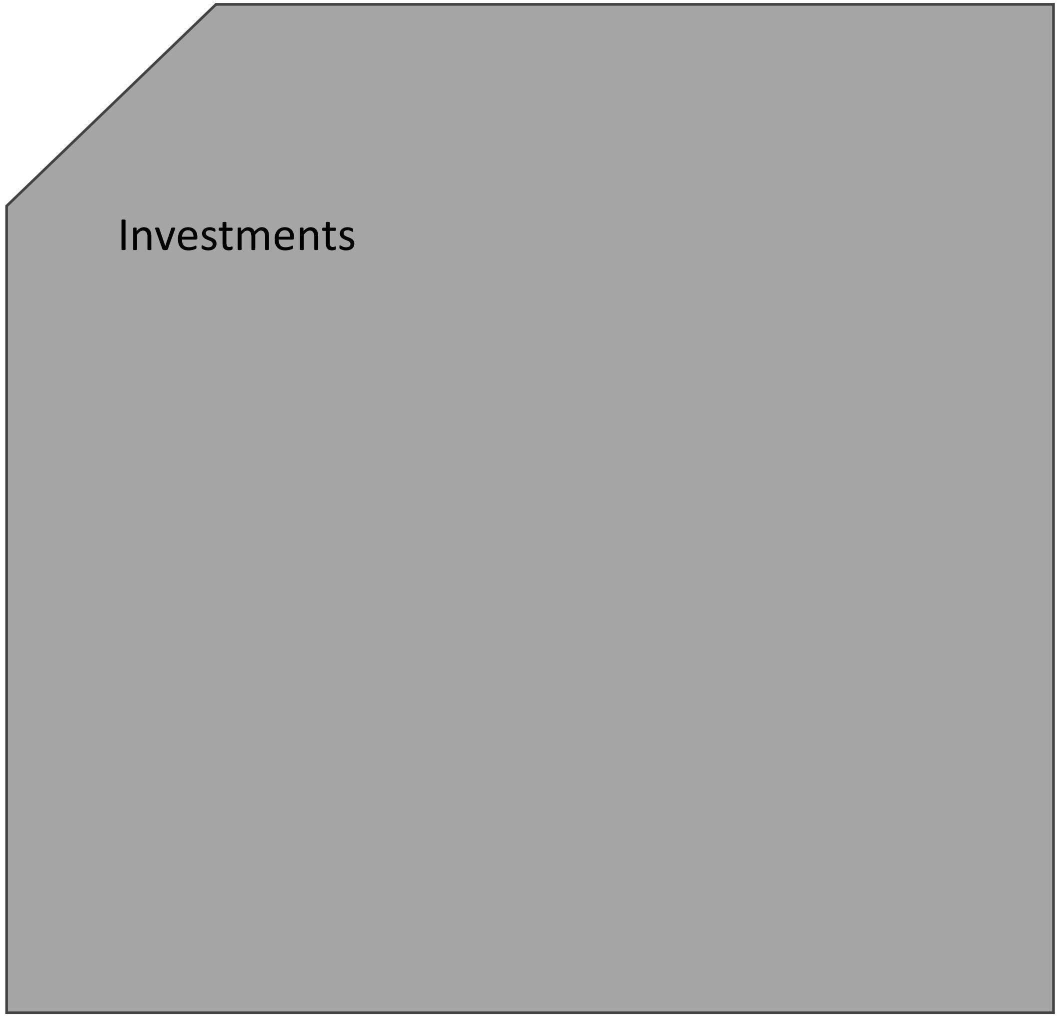
Our first project will be a (simple) application to manage investments in a cryptocurrency portfolio

What kinds of data do we need to store?
(yes, I know data is worthless but it still deserves to be stored)

Let’s Build an App!

Our first project will be a (simple) application to manage investments in a cryptocurrency portfolio

What kinds of data do we need to store?
(yes, I know data is worthless but it still deserves to be stored)



Let’s Build an App!

Our first project will be a (simple) application to manage investments in a cryptocurrency portfolio

What kinds of data do we need to store?
(yes, I know data is worthless but it still deserves to be stored)

Investments

- name of the coin

Let’s Build an App!

Our first project will be a (simple) application to manage investments in a cryptocurrency portfolio

What kinds of data do we need to store?
(yes, I know data is worthless but it still deserves to be stored)

Investments

- name of the coin
- quantity being purchased

Let’s Build an App!

Our first project will be a (simple) application to manage investments in a cryptocurrency portfolio

What kinds of data do we need to store?
(yes, I know data is worthless but it still deserves to be stored)

Investments

- name of the coin
- quantity being purchased
- buy or sell

Let's Build an App!

Our first project will be a (simple) application to manage investments in a cryptocurrency portfolio

What kinds of data do we need to store?
(yes, I know data is worthless but it still deserves to be stored)

Investments

- name of the coin
- quantity being purchased
- buy or sell
- timestamp

Let’s Build an App!

Our first project will be a (simple) application to manage investments in a cryptocurrency portfolio

What kinds of data do we need to store?
(yes, I know data is worthless but it still deserves to be stored)

Investments

- name of the coin
- quantity being purchased
- buy or sell
- timestamp

Portfolio

Let's Build an App!

Our first project will be a (simple) application to manage investments in a cryptocurrency portfolio

What kinds of data do we need to store?
(yes, I know data is worthless but it still deserves to be stored)

Investments

- name of the coin (*str*)
- quantity being purchased
- buy or sell
- timestamp

Portfolio

Let’s Build an App!

Our first project will be a (simple) application to manage investments in a cryptocurrency portfolio

What kinds of data do we need to store?
(yes, I know data is worthless but it still deserves to be stored)

Investments

- name of the coin (*str*)
- quantity being purchased (*float*)
- buy or sell
- timestamp

Portfolio

Let’s Build an App!

Our first project will be a (simple) application to manage investments in a cryptocurrency portfolio

What kinds of data do we need to store?
(yes, I know data is worthless but it still deserves to be stored)

Investments

- name of the coin (*str*)
- quantity being purchased (*float*)
- buy or sell (*bool*)
- timestamp

Portfolio

Let's Build an App!

Our first project will be a (simple) application to manage investments in a cryptocurrency portfolio

What kinds of data do we need to store?
(yes, I know data is worthless but it still deserves to be stored)

Investments

- name of the coin (*str*)
- quantity being purchased (*float*)
- buy or sell (*bool*)
- timestamp (*???*)

Portfolio

Let’s Build an App!

Our first project will be a (simple) application to manage investments in a cryptocurrency portfolio

What kinds of data do we need to store?
(yes, I know data is worthless but it still deserves to be stored)

Investments (*dict*)

- name of the coin (*str*)
- quantity being purchased (*float*)
- buy or sell (*bool*)
- timestamp (*???*)

Portfolio

Let’s Build an App!

Our first project will be a (simple) application to manage investments in a cryptocurrency portfolio

What kinds of data do we need to store?
(yes, I know data is worthless but it still deserves to be stored)

Investments (*dict*)

- name of the coin (*str*)
- quantity being purchased (*float*)
- buy or sell (*bool*)
- timestamp (*???*)

Portfolio (*list*)

Let's Build an App!

Our first project will be a (simple) application to manage investments in a cryptocurrency portfolio

What kinds of data do we need to store?
(yes, I know data is worthless but it still deserves to be stored)

Investments (*dict*)

- name of the coin (*str*)
- quantity being purchased (*float*)
- buy or sell (*bool*)
- timestamp (*???*)

Portfolio (*list*)

Summarize the portfolio with the total owned of each coin

Your turn!

Print the name and quantity of each investment in the portfolio

Extra credit: Print a message for each investment with the name, quantity and if it is a buy or sell (use the words “buy” and “sell” in the message)

Functions

Functions

In Python, a function is a named block of code

Functions

In Python, a function is a named block of code

To create a function, use the def keyword

Functions

In Python, a function is a named block of code

To create a function, use the def keyword

```
def my_func
```

Functions

In Python, a function is a named block of code

To create a function, use the def keyword

```
def my_func
```

After the function name, come the parameters

Functions

In Python, a function is a named block of code

To create a function, use the def keyword

```
def my_func
```

After the function name, come the parameters

```
def my_func(param1, param2)
```

Functions

In Python, a function is a named block of code

To create a function, use the def keyword

```
def my_func
```

After the function name, come the parameters

```
def my_func(param1, param2)
```

Followed by a colon

Functions

In Python, a function is a named block of code

To create a function, use the def keyword

```
def my_func
```

After the function name, come the parameters

```
def my_func(param1, param2)
```

Followed by a colon

```
def my_func(param1, param2):
```

Functions

In Python, a function is a named block of code

To create a function, use the def keyword

```
def my_func
```

After the function name, come the parameters

```
def my_func(param1, param2)
```

Followed by a colon

```
def my_func(param1, param2):
```

The body of the function is indented

Functions

In Python, a function is a named block of code

To create a function, use the def keyword

```
def my_func
```

After the function name, come the parameters

```
def my_func(param1, param2)
```

Followed by a colon

```
def my_func(param1, param2):
```

The body of the function is indented

```
def my_func(param1, param2):
  print(f"{param1}, {param2}")
```

Functions

In Python, a function is a named block of code

To create a function, use the def keyword

```
def my_func
```

After the function name, come the parameters

```
def my_func(param1, param2)
```

Followed by a colon

```
def my_func(param1, param2):
```

The body of the function is indented

```
def my_func(param1, param2):
  print(f"{param1}, {param2}")
```

Call the function by name, passing it the parameters

Functions

In Python, a function is a named block of code

To create a function, use the def keyword

```
def my_func
```

After the function name, come the parameters

```
def my_func(param1, param2)
```

Followed by a colon

```
def my_func(param1, param2):
```

The body of the function is indented

```
def my_func(param1, param2):
  print(f”{param1}, {param2}”)
```

Call the function by name, passing it the parameters

```
my_func(“Hello”, “World”)
```

Functions

In Python, a function is a named block of code

To create a function, use the def keyword

```
def my_func
```

After the function name, come the parameters

```
def my_func(param1, param2)
```

Followed by a colon

```
def my_func(param1, param2):
```

The body of the function is indented

```
def my_func(param1, param2):
  print(f"{param1}, {param2}")
```

Call the function by name, passing it the parameters

```
my_func("Hello", "World")
```

Trailing parameters can have default values

Functions

In Python, a function is a named block of code

To create a function, use the def keyword

```
def my_func
```

After the function name, come the parameters

```
def my_func(param1, param2)
```

Followed by a colon

```
def my_func(param1, param2):
```

The body of the function is indented

```
def my_func(param1, param2):
  print(f”{param1}, {param2}”)
```

Call the function by name, passing it the parameters

```
my_func(“Hello”, “World”)
```

Trailing parameters can have default values

```
def my_func(param1, param2=“world”):
  print(f”{param1}, {param2}”)
```

Functions

In Python, a function is a named block of code

To create a function, use the def keyword

```
def my_func
```

After the function name, come the parameters

```
def my_func(param1, param2)
```

Followed by a colon

```
def my_func(param1, param2):
```

The body of the function is indented

```
def my_func(param1, param2):
  print(f"{param1}, {param2}")
```

Call the function by name, passing it the parameters

```
my_func("Hello", "World")
```

Trailing parameters can have default values

```
def my_func(param1, param2="world"):
  print(f"{param1}, {param2}")
```

And can be omitted when function is called

Functions

In Python, a function is a named block of code

To create a function, use the def keyword

```
def my_func
```

After the function name, come the parameters

```
def my_func(param1, param2)
```

Followed by a colon

```
def my_func(param1, param2):
```

The body of the function is indented

```
def my_func(param1, param2):
  print(f”{param1}, {param2}”)
```

Call the function by name, passing it the parameters

```
my_func(“Hello”, “World”)
```

Trailing parameters can have default values

```
def my_func(param1, param2=“world”):
  print(f”{param1}, {param2}”)
```

And can be omitted when function is called

```
my_func(“Hello”)
```

Implement a function to add an investment to the portfolio.

The behavior of the application should not change.

Imports

Imports

Code may be organized into modules (namespaces)

Imports

Code may be organized into modules (namespaces)

To access code in a module, it must be imported

Imports

Code may be organized into modules (namespaces)

To access code in a module, it must be imported

The Python Standard Library includes modules that implement common tasks, such as handling dates

Imports

Code may be organized into modules (namespaces)

To access code in a module, it must be imported

The Python Standard Library includes modules that implement common tasks, such as handling dates

```
import datetime
```

Imports

Code may be organized into modules (namespaces)

To access code in a module, it must be imported

The Python Standard Library includes modules that implement common tasks, such as handling dates

```
import datetime
```

Members of a module must be prefixed by the module name

Imports

Code may be organized into modules (namespaces)

To access code in a module, it must be imported

The Python Standard Library includes modules that implement common tasks, such as handling dates

```
import datetime
```

Members of a module must be prefixed by the module name

```
today = datetime.datetime.now()
```

Imports

Code may be organized into modules (namespaces)

To access code in a module, it must be imported

The Python Standard Library includes modules that implement common tasks, such as handling dates

Members of a module must be prefixed by the module name

```
import datetime
```

```
today = datetime.datetime.now()
```

module name

Imports

Code may be organized into modules (namespaces)

To access code in a module, it must be imported

The Python Standard Library includes modules that implement common tasks, such as handling dates

Members of a module must be prefixed by the module name

```
import datetime

today = datetime.datetime.now()
```

module member

Imports

Code may be organized into modules (namespaces)

To access code in a module, it must be imported

The Python Standard Library includes modules that implement common tasks, such as handling dates

Members of a module must be prefixed by the module name

A member may also be explicitly imported, and referenced without the containing module

```
import datetime

today = datetime.datetime.now()
```

Imports

Code may be organized into modules (namespaces)

To access code in a module, it must be imported

The Python Standard Library includes modules that implement common tasks, such as handling dates

```
import datetime
```

Members of a module must be prefixed by the module name

```
today = datetime.datetime.now()
```

A member may also be explicitly imported, and referenced without the containing module

```
from datetime import datetime
```

Imports

Code may be organized into modules (namespaces)

To access code in a module, it must be imported

The Python Standard Library includes modules that implement common tasks, such as handling dates

Members of a module must be prefixed by the module name

A member may also be explicitly imported, and referenced without the containing module

```
import datetime

today = datet                )

from datetime import datetime
```

module name

Imports

Code may be organized into modules (namespaces)

To access code in a module, it must be imported

The Python Standard Library includes modules that implement common tasks, such as handling dates

Members of a module must be prefixed by the module name

A member may also be explicitly imported, and referenced without the containing module

```
import datetime

today  module member me.now()

from datetime import datetime
```

Imports

Code may be organized into modules (namespaces)

To access code in a module, it must be imported

The Python Standard Library includes modules that implement common tasks, such as handling dates

```
import datetime
```

Members of a module must be prefixed by the module name

```
today = datetime.datetime.now()
```

A member may also be explicitly imported, and referenced without the containing module

```
from datetime import datetime
today = datetime.now()
```

Imports

Code may be organized into modules (namespaces)

To access code in a module, it must be imported

The Python Standard Library includes modules that implement common tasks, such as handling dates

```
import datetime
```

Members of a module must be prefixed by the module name

```
today = datetime.datetime.now()
```

A member may also be explicitly imported, and referenced without the containing module

```
from datetime import datetime
today = datetime.now()
```

Imports can have aliases

Imports

Code may be organized into modules (namespaces)

To access code in a module, it must be imported

The Python Standard Library includes modules that implement common tasks, such as handling dates

```
import datetime
```

Members of a module must be prefixed by the module name

```
today = datetime.datetime.now()
```

A member may also be explicitly imported, and referenced without the containing module

```
from datetime import datetime
today = datetime.now()
```

Imports can have aliases

```
import datetime as dt
today = dt.datetime.now()
```

Imports

Code may be organized into modules (namespaces)

To access code in a module, it must be imported

The Python Standard Library includes modules that implement common tasks, such as handling dates

```
import datetime
```

Members of a module must be prefixed by the module name

```
today = datetime.datetime.now()
```

A member may also be explicitly imported, and referenced without the containing module

```
from datetime import datetime
today = datetime.now()
```

Imports can have aliases

```
import datetime as dt
today = dt.datetime.now()
```

All members can be imported at the same time

Imports

Code may be organized into modules (namespaces)

To access code in a module, it must be imported

The Python Standard Library includes modules that implement common tasks, such as handling dates

```
import datetime
```

Members of a module must be prefixed by the module name

```
today = datetime.datetime.now()
```

A member may also be explicitly imported, and referenced without the containing module

```
from datetime import datetime
today = datetime.now()
```

Imports can have aliases

```
import datetime as dt
today = dt.datetime.now()
```

All members can be imported at the same time

```
from datetime import *
```

Imports

Code may be organized into modules (namespaces)

To access code in a module, it must be imported

The Python Standard Library includes modules that implement common tasks, such as handling dates

```
import datetime
```

Members of a module must be prefixed by the module name

```
today = datetime.datetime.now()
```

A member may also be explicitly imported, and referenced without the containing module

```
from datetime import datetime
today = datetime.now()
```

Imports can have aliases

```
import datetime as dt
today = dt.datetime.now()
```

Never do this!

All members can be imported at the same time

```
from datetime import *
```

Your turn!

Add the timestamp to each investment

Extra credit: include the timestamp in the investment description (hint: look up strftime in the Python documentation)

The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

NumPy

data is worthless

Another look at lists

Suppose we wanted to perform an operator on every value in a list.

Another look at lists

Suppose we wanted to perform an operator on every value in a list.

```
my_list = [2, 3, 5, 7, 11, 13]
```

Another look at lists

Suppose we wanted to perform an operator on every value in a list.

Will this code work?

```
my_list = [2, 3, 5, 7, 11, 13]
```

Another look at lists

Suppose we wanted to perform an operator on every value in a list.

```
my_list = [2, 3, 5, 7, 11, 13]
```

Will this code work?

```
my_list + 1
```

Another look at lists

Suppose we wanted to perform an operator on every value in a list.

```
my_list = [2, 3, 5, 7, 11, 13]
```

Will this code work?

```
my_list + 1
```

The error tells us two things:

1. The + operator will concatenate two lists
2. You can’t concatenate a list and an int

Another look at lists

Suppose we wanted to perform an operator on every value in a list.

```
my_list = [2, 3, 5, 7, 11, 13]
```

Will this code work?

```
my_list + 1
```

The error tells us two things:

1. The + operator will concatenate two lists
2. You can't concatenate a list and an int

Try concatenating two lists

Another look at lists

Suppose we wanted to perform an operator on every value in a list.

```
my_list = [2, 3, 5, 7, 11, 13]
```

Will this code work?

```
my_list + 1
```

The error tells us two things:

1. The + operator will concatenate two lists
2. You can’t concatenate a list and an int

Try concatenating two lists

```
my_other_list = [1, 2, 3, 4, 5, 6]
my_list + my_other_list
```

Another look at lists

Suppose we wanted to perform an operator on every value in a list.

```
my_list = [2, 3, 5, 7, 11, 13]
```

Will this code work?

```
my_list + 1
```

The error tells us two things:

1. The + operator will concatenate two lists
2. You can't concatenate a list and an int

Try concatenating two lists

```
my_other_list = [1, 2, 3, 4, 5, 6]
my_list + my_other_list
```

How can we add 1 to every value in a list?

Another look at lists

Suppose we wanted to perform an operator on every value in a list.

```
my_list = [2, 3, 5, 7, 11, 13]
```

Will this code work?

```
my_list + 1
```

The error tells us two things:

1. The + operator will concatenate two lists
2. You can’t concatenate a list and an int

Try concatenating two lists

```
my_other_list = [1, 2, 3, 4, 5, 6]
my_list + my_other_list
```

How can we add 1 to every value in a list?

```
[value + 1 for value in my_list]
```

Commercial break: list comprehensions

Very often in Python, we need to iterate over a list and do somethine with the elements. The obligatory example is squaring the first 10 integers.

Commercial break: list comprehensions

Very often in Python, we need to iterate over a list and do somethine with the elements. The obligatory example is squaring the first 10 integers.

```
squares = []
for value in list(range(1, 11)):
  squares.append(value ** 2)
```

Commercial break: list comprehensions

Very often in Python, we need to iterate over a list and do somethine with the elements. The obligatory example is squaring the first 10 integers.

```
squares = []
for value in list(range(1, 11)):
  squares.append(value ** 2)
```

This works. It's legal Python code. But this is done so often, that Python offers a shortcut, the list comprehension.

Commercial break: list comprehensions

Very often in Python, we need to iterate over a list and do somethine with the elements. The obligatory example is squaring the first 10 integers.

```
squares = []
for value in list(range(1, 11)):
  squares.append(value ** 2)
```

This works. It's legal Python code. But this is done so often, that Python offers a shortcut, the list comprehension.

```
squares = [
  value ** 2 for value in list(range(1, 11))
]
```

Commercial break: list comprehensions

Very often in Python, we need to iterate over a list and do somethine with the elements. The obligatory example is squaring the first 10 integers.

```
squares = []
for value in list(range(1, 11)):
  squares.append(value ** 2)
```

This works. It's legal Python code. But this is done so often, that Python offers a shortcut, the list comprehension.

```
squares = [
  value ** 2 for value in list(range(1, 11))
]
```

This says that Python will create a temporary list. Then for each element in the source, it will perform an action and place the result in the temporary list. After the source is exhausted, the temporary list is returned.

Commercial break: list comprehensions

Very often in Python, we need to iterate over a list and do somethine with the elements. The obligatory example is squaring the first 10 integers.

```
squares = []
for value in list(range(1, 11)):
  squares.append(value ** 2)
```

This works. It's legal Python code. But this is done so often, that Python offers a shortcut, the list comprehension.

```
squares = [
  value ** 2 for value in list(range(1, 11))
]
```

This says that Python will create a temporary list. Then for each element in the source, it will perform an action and place the result in the temporary list. After the source is exhausted, the temporary list is returned.

You can also add a conditional to filter the elements in the source.

Commercial break: list comprehensions

Very often in Python, we need to iterate over a list and do somethine with the elements. The obligatory example is squaring the first 10 integers.

```
squares = []
for value in list(range(1, 11)):
  squares.append(value ** 2)
```

This works. It's legal Python code. But this is done so often, that Python offers a shortcut, the list comprehension.

```
squares = [
  value ** 2 for value in list(range(1, 11))
]
```

This says that Python will create a temporary list. Then for each element in the source, it will perform an action and place the result in the temporary list. After the source is exhausted, the temporary list is returned.

You can also add a conditional to filter the elements in the source.

```
even_squares = [
  value ** 2 for value in list(range(1, 11))
  if value % 2 == 0
]
```

Another look at lists

Suppose we wanted to perform an operator on every value in a list.

```
my_list = [2, 3, 5, 7, 11, 13]
```

Will this code work?

```
my_list + 1
```

The error tells us two things:

1. The + operator will concatenate two lists
2. You can't concatenate a list and an int

Try concatenating two lists

```
my_other_list = [1, 2, 3, 4, 5, 6]
my_list + my_other_list
```

How can we add 1 to every value in a list?

```
[value + 1 for value in my_list]
```

How can we perform an elementwise addition between two lists?

Another look at lists

Suppose we wanted to perform an operator on every value in a list.

```
my_list = [2, 3, 5, 7, 11, 13]
```

Will this code work?

```
my_list + 1
```

The error tells us two things:

1. The + operator will concatenate two lists
2. You can't concatenate a list and an int

Try concatenating two lists

```
my_other_list = [1, 2, 3, 4, 5, 6]
my_list + my_other_list
```

How can we add 1 to every value in a list?

```
[value + 1 for value in my_list]
```

How can we perform an elementwise addition between two lists?

```
[
  sum(pair)
  for pair in zip(my_list, my_other_list)
]
```

Another look at lists

Suppose we wanted to perform an operator on every value in a list.

```
my_list = [2, 3, 5, 7, 11, 13]
```

Will this code work?

```
my_list + 1
```

The error tells us two things:

1. The + operator will concatenate two lists
2. You can’t concatenate a list and an int

Try concatenating two lists

```
my_other_list = [1, 2, 3, 4, 5, 6]
my_list + my_other_list
```

How can we add 1 to every value in a list?

```
[value + 1 for value in my_list]
```

How can we perform an elementwise addition between two lists?

```
[
  sum(pair)
  for pair in zip(my_list, my_other_list)
]
```

What rule of The Zen of Python does this violate?

NumPy

A Python package for working with big globs of numbers

NumPy

A Python package for working with big globs of numbers

To use the NumPy package, you must import the numpy module

NumPy

A Python package for working with big globs of numbers

To use the NumPy package, you must import the numpy module

```
import numpy as np
```

NumPy

A Python package for working with big globs of numbers

To use the NumPy package, you must import the numpy module

```
import numpy as np
```

Notice that the numpy module is aliased as np. Data scientists are lazy. They don't want to type a lot.

NumPy

A Python package for working with big globs of numbers

To use the NumPy package, you must import the numpy module

```
import numpy as np
```

Notice that the numpy module is aliased as np. Data scientists are lazy. They don't want to type a lot.

The fundamental data structure in NumPy is the array.

NumPy

A Python package for working with big globs of numbers

To use the NumPy package, you must import the numpy module

```
import numpy as np
```

Notice that the numpy module is aliased as np. Data scientists are lazy. They don't want to type a lot.

The fundamental data structure in NumPy is the array.

You can create a NumPy array from a Python list using the array function.

NumPy

A Python package for working with big globs of numbers

To use the NumPy package, you must import the numpy module

```
import numpy as np
```

Notice that the numpy module is aliased as np. Data scientists are lazy. They don't want to type a lot.

The fundamental data structure in NumPy is the array.

You can create a NumPy array from a Python list using the array function.

```
my_array = np.array(my_list)
```

NumPy

A NumPy array is like a Python list.

NumPy

A NumPy array is like a Python list.

You can get the length of a NumPy array

NumPy

A NumPy array is like a Python list.

You can get the length of a NumPy array

```
len(my_array)
```

NumPy

A NumPy array is like a Python list.

You can get the length of a NumPy array

```
len(my_array)
```

You can index into a NumPy array

NumPy

A NumPy array is like a Python list.

You can get the length of a NumPy array

```
len(my_array)
```

You can index into a NumPy array

```
my_array[2]
```

NumPy

A NumPy array is like a Python list.

You can get the length of a NumPy array

```
len(my_array)
```

You can index into a NumPy array

```
my_array[2]
```

You can iterate over a NumPy array in a for loop

NumPy

A NumPy array is like a Python list.

You can get the length of a NumPy array

```
len(my_array)
```

You can index into a NumPy array

```
my_array[2]
```

You can iterate over a NumPy array in a for loop

```
for item in my_array:
  print(item)
```

NumPy

A NumPy array is like a Python list.

You can get the length of a NumPy array

```
len(my_array)
```

You can index into a NumPy array

```
my_array[2]
```

You can iterate over a NumPy array in a for loop

```
for item in my_array:
  print(item)
```

You can slice a NumPy array.

NumPy

A NumPy array is like a Python list.

You can get the length of a NumPy array

```
len(my_array)
```

You can index into a NumPy array

```
my_array[2]
```

You can iterate over a NumPy array in a for loop

```
for item in my_array:
  print(item)
```

You can slice a NumPy array.

```
my_slice = my_array[3:8]
```

NumPy

A NumPy array is like a Python list.

You can get the length of a NumPy array

```
len(my_array)
```

You can index into a NumPy array

```
my_array[2]
```

You can iterate over a NumPy array in a for loop

```
for item in my_array:
    print(item)
```

You can slice a NumPy array.

```
my_slice = my_array[3:8]
```

Note that slicing a NumPy array returns a view and modifying the view modifies the array.

NumPy

A NumPy array is like a Python list.

You can get the length of a NumPy array

```
len(my_array)
```

You can index into a NumPy array

```
my_array[2]
```

You can iterate over a NumPy array in a for loop

```
for item in my_array:
  print(item)
```

You can slice a NumPy array.

```
my_slice = my_array[3:8]
```

Note that slicing a NumPy array returns a view and modifying the view modifies the array.

```
my_slice[0] = 1000
```

NumPy

A NumPy array is like a Python list.

You can get the length of a NumPy array

```
len(my_array)
```

You can index into a NumPy array

```
my_array[2]
```

You can iterate over a NumPy array in a for loop

```
for item in my_array:
    print(item)
```

You can slice a NumPy array.

```
my_slice = my_array[3:8]
```

Note that slicing a NumPy array returns a view and modifying the view modifies the array.

```
my_slice[0] = 1000
```

But with a Python list, a slice is a copy of the data. Modifying the copy does not modify the list

NumPy

A NumPy array is like a Python list.

You can get the length of a NumPy array

```
len(my_array)
```

You can index into a NumPy array

```
my_array[2]
```

You can iterate over a NumPy array in a for loop

```
for item in my_array:
  print(item)
```

You can slice a NumPy array.

```
my_slice = my_array[3:8]
```

Note that slicing a NumPy array returns a view and modifying the view modifies the array.

```
my_slice[0] = 1000
```

But with a Python list, a slice is a copy of the data. Modifying the copy does not modify the list

```
my_slice = my_list[3:8]
my_slice[0] = 1000
```

NumPy Array Superpowers

Try adding 1 to a NumPy array

NumPy Array Superpowers

Try adding 1 to a NumPy array

```
my_array + 1
```

NumPy Array Superpowers

Try adding 1 to a NumPy array

This also works with subtraction

```
my_array + 1
```

NumPy Array Superpowers

Try adding 1 to a NumPy array

```
my_array + 1
```

This also works with subtraction

```
my_array - 1
```

NumPy Array Superpowers

Try adding 1 to a NumPy array

```
my_array + 1
```

This also works with subtraction

```
my_array - 1
```

You can also add two NumPy arrays

NumPy Array Superpowers

Try adding 1 to a NumPy array

```
my_array + 1
```

This also works with subtraction

```
my_array - 1
```

You can also add two NumPy arrays

```
my_other_array = np.array(my_other_list)
my_array + my_other_array
```

NumPy Array Superpowers

Try adding 1 to a NumPy array

```
my_array + 1
```

This also works with subtraction

```
my_array - 1
```

You can also add two NumPy arrays

```
my_other_array = np.array(my_other_list)
my_array + my_other_array
```

This functionality is called broadcasting. It's the first NumPy array superpower.

NumPy Array Superpowers

Try adding 1 to a NumPy array

```
my_array + 1
```

This also works with subtraction

```
my_array - 1
```

You can also add two NumPy arrays

```
my_other_array = np.array(my_other_list)
my_array + my_other_array
```

This functionality is called broadcasting. It's the first NumPy array superpower.

You can only do an elementwise addition on NumPy arrays of the same length

NumPy Array Superpowers

Try adding 1 to a NumPy array

```
my_array + 1
```

This also works with subtraction

```
my_array - 1
```

You can also add two NumPy arrays

```
my_other_array = np.array(my_other_list)
my_array + my_other_array
```

This functionality is called broadcasting. It's the first NumPy array superpower.

You can only do an elementwise addition on NumPy arrays of the same length

```
my_array + my_other_array[:-1]
```

NumPy Array Superpowers

Try adding 1 to a NumPy array

```
my_array + 1
```

This also works with subtraction

```
my_array - 1
```

You can also add two NumPy arrays

```
my_other_array = np.array(my_other_list)
my_array + my_other_array
```

This functionality is called broadcasting. It's the first NumPy array superpower.

You can only do an elementwise addition on NumPy arrays of the same length

```
my_array + my_other_array[:-1]
```

Note that the error message refers to the lengths as the shapes of the arrays. That brings to the second NumPy array superpower.

NumPy Array Superpowers

The second NumPy array superpower is multi dimensionality

You can represent a multi dimensional array using a nested Python list

```
my_list = [[2, 3, 5], [7, 11, 13]]
```

And use the NumPy array function to turn it into an array

```
my_array = np.array(my_list)
```

The array has a shape property which is a tuple with the size of each dimension

```
my_array.shape
```

And a size property that counts the total number of values in the array

```
my_array.size
```

The array size is the product of the size of the dimensions in the array

Commercial Break: Random Numbers

To reduce typing, we will generate random NumPy array for a little while

NumPy uses a random number generator object which can be created with the default_rng function

```
rng = np.random.default_rng()
```

The integers method will return a random array of integers with a lower and upper (ex.) bound and size

```
my_array = rng.integers(0, 10, (5, 4,))
```

So everyone is on the same page, we can seed the generator so it produces the same sequence

```
seed_rng = np.random.default_rng(3824)
```

And use the integers method again

```
my_array = seed_rng.integers(0, 10, (5, 4,))
```

Rerun the cells with the seeded and non-seeded generators. The seeded generator always returns the same sequence of numbers instead of random.

NumPy Array Superpowers

You can reshape a NumPy array

```
my_array.reshape((10, 2,))
```

You can add (or remove) dimensions

```
my_array.reshape((2, 5, 2,))
```

The product of the sizes of the old and new dimensions must be the same

```
my_array.reshape((5, 2,))
```

And you can flatten the array

```
my_array.flatten()
```

Note these methods do not modify the shape of the calling array. To do that, you can assign a new value to the shape property

```
my_array.shape = (10, 2,)
```

NumPy Array Superpowers

The third NumPy array superpower is special syntax for indexing and slicing

Create an array with lots of dimensions.

```
my_array = seed_rng.integers(0, 10,
  (4, 2, 3, 4,))
```

Index into the array to get a single value.

```
my_array[1][0][2][1]
```

The inner square brackets create syntactic noise so NumPy allows you to replace them with commas

```
my_array[1, 0, 2, 1]
```

NumPy Array Superpowers

Recall a slice that omits both bounds will return a copy of the array

```
my_array = seed_rng.integers(0, 10, (5, 4,))
my_array[:]
```

Indexing into the copy will return …

```
my_array[:][1]
```

Using the shorthand syntax with commas instead of square brackets

```
my_array[:, 1]
```

The result is a single dimension NumPy array with length 5 …

And my_array has 5 rows …

Each column has 5 values

The result is the column at index 1 as a NumPy array

Extra Credit

How would you slice the array to return only the inner elements?

Extra Credit

How would you slice the array to return only the inner elements?

```
array([[4, 1],
       [4, 5],
       [3, 8]])
```

Extra Credit

How would you slice the array to return only the inner elements?

```
array([[4, 1],
       [4, 5],
       [3, 8]])
```

```
my_array[1:4, 1:3]
```

Extra Credit

How would you slice the array to return only the inner elements?

```
array([[4, 1],
       [4, 5],
       [3, 8]])
```

```
my_array[1:4, 1:3]
```

Can you write a function to generalize it for any 2D array?

Extra Credit

How would you slice the array to return only the inner elements?

```
array([[4, 1],
       [4, 5],
       [3, 8]])
```

```
my_array[1:4, 1:3]
```

Can you write a function to generalize it for any 2D array?

```
def get_inner(a):
    return a[1:a.shape[0]-1, 1:a.shape[1]-1]
```

NumPy Array Superpowers

NumPy includes a number of functions for things such as summary statistics, i.e. median

```
np.median(my_array)
```

Some functions, such as sum, have equivalents in Python. But you should always use the NumPy versions with arrays.

```
np.sum(my_array)
sum(my_array)
```

Same with min and max

```
np.max(my_array)
```

NumPy also includes the argmin and argmax functions that return the index of the minimum of maximum value. (this will be revisited later)

```
np.argmax(my_array)
```

Many of these functions are implemented as methods on the array (breaking the Zen of Python)

```
my_array.argmax()
```

pandas

analytics are worth pennies

pandas

The first fundamental data structure in pandas is the Series

pandas

The first fundamental data structure in pandas is the Series

A Series in pandas is like a Python list, with an associated index

pandas

The first fundamental data structure in pandas is the Series

A Series in pandas is like a Python list, with an associated index

You can create a Series from a Python list or a NumPy array

pandas

The first fundamental data structure in pandas is the Series

A Series in pandas is like a Python list, with an associated index

You can create a Series from a Python list or a NumPy array

```
my_series = pd.Series(my_list)
my_series = pd.Series(my_array)
```

pandas

The first fundamental data structure in pandas is the Series

A Series in pandas is like a Python list, with an associated index

You can create a Series from a Python list or a NumPy array

```
my_series = pd.Series(my_list)
my_series = pd.Series(my_array)
```

Notice the Series has a dtype or data type

pandas

The first fundamental data structure in pandas is the Series

A Series in pandas is like a Python list, with an associated index

You can create a Series from a Python list or a NumPy array

```
my_series = pd.Series(my_list)
my_series = pd.Series(my_array)
```

Notice the Series has a dtype or data type

The values of a Series are a NumPy array

pandas

The first fundamental data structure in pandas is the Series

A Series in pandas is like a Python list, with an associated index

You can create a Series from a Python list or a NumPy array

```
my_series = pd.Series(my_list)
my_series = pd.Series(my_array)
```

Notice the Series has a dtype or data type

The values of a Series are a NumPy array

```
my_series.values
```

Commercial Break: faker

The faker package will generate random data used for testing and exploring.

Commercial Break: faker

The faker package will generate random data used for testing and exploring.

It is not part of the Python language or standard library so you must install it with pip

Commercial Break: faker

The faker package will generate random data used for testing and exploring.

It is not part of the Python language or standard library so you must install it with pip

```
%pip install faker
```

Commercial Break: faker

The faker package will generate random data used for testing and exploring.

It is not part of the Python language or standard library so you must install it with pip

```
%pip install faker
```

Import the Faker class from the faker module

Commercial Break: faker

The faker package will generate random data used for testing and exploring.

It is not part of the Python language or standard library so you must install it with pip

```
%pip install faker
```

Import the Faker class from the faker module

```
from faker import Faker
```

Commercial Break: faker

The faker package will generate random data used for testing and exploring.

It is not part of the Python language or standard library so you must install it with pip

```
%pip install faker
```

Import the Faker class from the faker module

```
from faker import Faker
```

Create and seed new instance of the Faker class

Commercial Break: faker

The faker package will generate random data used for testing and exploring.

It is not part of the Python language or standard library so you must install it with pip

```
%pip install faker
```

Import the Faker class from the faker module

```
from faker import Faker
```

Create and seed new instance of the Faker class

```
fake = Faker()
fake.seed_instance(3824)
```

Commercial Break: faker

The faker package will generate random data used for testing and exploring.

It is not part of the Python language or standard library so you must install it with pip

```
%pip install faker
```

Import the Faker class from the faker module

```
from faker import Faker
```

Create and seed new instance of the Faker class

```
fake = Faker()
fake.seed_instance(3824)
```

List the members of the Faker class with the built in dir function

Commercial Break: faker

The faker package will generate random data used for testing and exploring.

It is not part of the Python language or standard library so you must install it with pip

```
%pip install faker
```

Import the Faker class from the faker module

```
from faker import Faker
```

Create and seed new instance of the Faker class

```
fake = Faker()
fake.seed_instance(3824)
```

List the members of the Faker class with the built in dir function

```
dir(fake)
```

Commercial Break: faker

The faker package will generate random data used for testing and exploring.

It is not part of the Python language or standard library so you must install it with pip

```
%pip install faker
```

Import the Faker class from the faker module

```
from faker import Faker
```

Create and seed new instance of the Faker class

```
fake = Faker()
fake.seed_instance(3824)
```

List the members of the Faker class with the built in dir function

```
dir(fake)
```

Generate a list of random words

Commercial Break: faker

The faker package will generate random data used for testing and exploring.

It is not part of the Python language or standard library so you must install it with pip

```
%pip install faker
```

Import the Faker class from the faker module

```
from faker import Faker
```

Create and seed new instance of the Faker class

```
fake = Faker()
fake.seed_instance(3824)
```

List the members of the Faker class with the built in dir function

```
dir(fake)
```

Generate a list of random words

```
names = [fake.name() for _ in range(20)]
```

Commercial Break: faker

The faker package will generate random data used for testing and exploring.

It is not part of the Python language or standard library so you must install it with pip

```
%pip install faker
```

Import the Faker class from the faker module

```
from faker import Faker
```

Create and seed new instance of the Faker class

```
fake = Faker()
fake.seed_instance(3824)
```

List the members of the Faker class with the built in dir function

```
dir(fake)
```

Generate a list of random words

```
names = [fake.name() for _ in range(20)]
```

Generate a list from a set of values

Commercial Break: faker

The faker package will generate random data used for testing and exploring.

It is not part of the Python language or standard library so you must install it with pip

```
%pip install faker
```

Import the Faker class from the faker module

```
from faker import Faker
```

Create and seed new instance of the Faker class

```
fake = Faker()
fake.seed_instance(3824)
```

List the members of the Faker class with the built in dir function

```
dir(fake)
```

Generate a list of random words

```
names = [fake.name() for _ in range(20)]
```

Generate a list from a set of values

```
DEPTS = [“Sales”, “Admin”, “IT”, “Creative”]
depts = fake.random_choices(
  DEPTS, length=len(names))
```

pandas

Similar to a dictionary, you can access elements in a Series by the index

pandas

Similar to a dictionary, you can access elements in a Series by the index

```
name_series[2]
```

pandas

Similar to a dictionary, you can access elements in a Series by the index

The index does not have to be numeric

```
name_series[2]
```

pandas

Similar to a dictionary, you can access elements in a Series by the index

The index does not have to be numeric

```
name_series[2]

name_series.index = list(
  string.ascii_letters[:len(name_series)])
name_series[“f”]
```

pandas

Similar to a dictionary, you can access elements in a Series by the index

The index does not have to be numeric

Convert a Series to a dictionary

```
name_series[2]

name_series.index = list(
  string.ascii_letters[:len(name_series)])
name_series[“f”]
```

pandas

Similar to a dictionary, you can access elements in a Series by the index

```
name_series[2]
```

The index does not have to be numeric

```
name_series.index = list(
  string.ascii_letters[:len(name_series)])
name_series[“f”]
```

Convert a Series to a dictionary

```
name_series.to_dict()
```

pandas

Similar to a dictionary, you can access elements in a Series by the index

```
name_series[2]
```

The index does not have to be numeric

```
name_series.index = list(
  string.ascii_letters[:len(name_series)])
name_series[“f”]
```

Convert a Series to a dictionary

```
name_series.to_dict()
```

Write a Series to a JSON file

pandas

Similar to a dictionary, you can access elements in a Series by the index

```
name_series[2]
```

The index does not have to be numeric

```
name_series.index = list(
  string.ascii_letters[:len(name_series)])
name_series[“f”]
```

Convert a Series to a dictionary

```
name_series.to_dict()
```

Write a Series to a JSON file

```
name_series.to_json(“names.json”)
```

pandas

Similar to a dictionary, you can access elements in a Series by the index

```
name_series[2]
```

The index does not have to be numeric

```
name_series.index = list(
  string.ascii_letters[:len(name_series)])
name_series[“f”]
```

Convert a Series to a dictionary

```
name_series.to_dict()
```

Write a Series to a JSON file

```
name_series.to_json(“names.json”)
```

Read a Series from a JSON file

pandas

Similar to a dictionary, you can access elements in a Series by the index

```
name_series[2]
```

The index does not have to be numeric

```
name_series.index = list(
  string.ascii_letters[:len(name_series)])
name_series[“f”]
```

Convert a Series to a dictionary

```
name_series.to_dict()
```

Write a Series to a JSON file

```
name_series.to_json(“names.json”)
```

Read a Series from a JSON file

```
names =
  pd.read_json(“names.json”, typ=“series”)
```

pandas

Similar to a dictionary, you can access elements in a Series by the index

```
name_series[2]
```

The index does not have to be numeric

```
name_series.index = list(
  string.ascii_letters[:len(name_series)])
name_series[“f”]
```

Convert a Series to a dictionary

```
name_series.to_dict()
```

Write a Series to a JSON file

```
name_series.to_json(“names.json”)
```

Read a Series from a JSON file

```
names =
  pd.read_json(“names.json”, typ=“series”)
```

Generate a list of random words

pandas

Similar to a dictionary, you can access elements in a Series by the index

```
name_series[2]
```

The index does not have to be numeric

```
name_series.index = list(
  string.ascii_letters[:len(name_series)])
name_series[“f”]
```

Convert a Series to a dictionary

```
name_series.to_dict()
```

Write a Series to a JSON file

```
name_series.to_json(“names.json”)
```

Read a Series from a JSON file

```
names =
  pd.read_json(“names.json”, typ=“series”)
```

Generate a list of random words

```
words = [fake.word() for _ in range(10)]
```

pandas

Similar to a dictionary, you can access elements in a Series by the index

```
name_series[2]
```

The index does not have to be numeric

```
name_series.index = list(
  string.ascii_letters[:len(name_series)])
name_series[“f”]
```

Convert a Series to a dictionary

```
name_series.to_dict()
```

Write a Series to a JSON file

```
name_series.to_json(“names.json”)
```

Read a Series from a JSON file

```
names =
  pd.read_json(“names.json”, typ=“series”)
```

Generate a list of random words

```
words = [fake.word() for _ in range(10)]
```

Generate a list from a set of values

pandas

Similar to a dictionary, you can access elements in a Series by the index

```
name_series[2]
```

The index does not have to be numeric

```
name_series.index = list(
  string.ascii_letters[:len(name_series)])
name_series[“f”]
```

Convert a Series to a dictionary

```
name_series.to_dict()
```

Write a Series to a JSON file

```
name_series.to_json(“names.json”)
```

Read a Series from a JSON file

```
names =
  pd.read_json(“names.json”, typ=“series”)
```

Generate a list of random words

```
words = [fake.word() for _ in range(10)]
```

Generate a list from a set of values

```
SIZES = [“S”, “M”, “L”]
sizes = fake.random_choices(
  SIZES, length=10)
```

pandas

A Series can also use datetime objects for the index

```
sales = rng.integers(1, 10, (5,)) * 100
dates = [datetime.date(2024, 2, 1) ... ]
sales.index = dates
```

A better way in to generate a DatetimeIndex

```
dt_index = pd.date_range(
  “2024-02-01”, periods=5)
sales.index = dt.index
```

The default frequency is daily, but you could also generate datetimes hourly

```
pd.date_range(
  “2024-02-01”, periods=5, freq=“H”)
```

Or the first day of the month

```
pd.date_range(
  “2024-01-01”, periods=12, freq=“MS”)
```

Or Monday of each week

```
pd.date_range(
  “2024-02-01”, periods=4, freq=“W-MON”)
```

Regardless it makes generating date indexes easier

```
jan_sales = rng.integers(1, 10, (31,)) * 100
jan_sales_series = pd.Series(
  jan_sales, index=pd.date_range(
    “2021-01-01”, periods=31)
```

pandas

The loc field gets a value by index	`jan_sales_series.loc[“2021-01-04”]`
The iloc field gets a value by position (zero-based)	`jan_sales_series.iloc[3]`
Both loc and iloc can be sliced (upper bound is inclusive)	`jan_sales_series.loc[ “2021-01-04”:”2021-01-10”]`
Apply a boolean operator to a Series to get a Series of bools with the results for each value	`jan_sales_series > 500`
That Series of bools can then be used to select the subset of the Series for each True value	`jan_sales_series[jan_sales_series > 500]`

pandas

The statistical method of the NumPy are also present on the Series object

```
jan_sales_series.mean()
jan_sales_series.median()
```

The unique method returns an array of values in the Series less duplicates

```
depts_series.unique()
```

The value_counts methods returns a Series with a tally of each value

```
depts_series.value_counts()
```

And a series can be sorted by the index or the values.

```
jan_sales_series.sort_values()
jan_sales_series.sort_index()
```

These methods, by default return a new Series but the inplace keyword argument, set to True, with modify the Series that is sorted.

```
jan_sales_series.sort_values(inplace=True)
```

Also, the default sort order is ascending.

```
jan_sales_series.sort_values(
  inplace=True, ascending=False)
```

pandas

A DataFrame is a collection of Series with a common index. It is the second fundamental data structure.

Many different data sources can be used for a Dataframe, like a dictionary.

```
sales_data = {
  “Quarter”: [1, 2, 3, 4],
  “North”: rng.integers(1, 10, (4,)) * 100,
  “South”: rng.integers(1, 10, (4,)) * 100,
  “East”: rng.integers(1, 10, (4,)) * 100,
  “West”: rng.integers(1, 10, (4,)) * 100
}
sales_df = pd.DataFrame(sales_data)
```

When displaying a DataFrame, Jupyter Notebook uses CSS to format it.

pandas

The columns fields is a collection (index) of the column names	`sales_df.columns`
A column's value can be accessed using dictionary-like syntax with square brackets	`sales_df[“North”]` `sales_df[“South”]`
A DataFrame will also create a dynamic field for each column name, though its use is discouraged.	`sales_df.North` `sales_df.South`
The values are stored in a 2D NumPy array	`sales_df.values`
The head method returns the first n rows (default 5)	`sales_df.head(2)`
The tail method returns the last n rows (default 5)	`sales_df.tail(2)`

pandas

The info method will return data about the columns, their datatypes and if they have null values

```
sales_df.info()
```

The describe method will return a DataFrame with statistics about numerical columns

```
sales_df.describe()
```

Create a new column by assigning a Series to the column name

```
sales_df[“Canada”] = pd.Series(
  rng.integers(1, 10, (3,)) * 100)
```

The column can be removed using the del keyword

```
del sales_df[“Canada”]
```

pandas

Sometimes values will be missing from a DataFrame

A Series can specify the index for which values are present

```
sales_df["South America"] = pd.Series(
  rng.integers(1, 10, (3,)) * 100,
  index=[0, 1, 3])
```

The isna method will return a DataFrame of bools

```
sales_df.isna()
```

The dropna method will remove data that contains missing values

```
sales_df.dropna()
```

The fillna method will replace missing values

```
sales_df.fillna(0)
```

pandas

Computing statistical functions is also possible	`sales_df.sum()`
And retrieving rows with loc and iloc	`sales_df.loc[1]`
Unlike a Series, loc and iloc support multiple dimensions	`sales_df.loc[1, “North”]`
And loc and iloc support slicing	`sales_df.loc[:, “North”:”West”]`
Write a DataFrame to a CSV	`sales_df.to_csv(“sales.csv”)`
Read a DataFrame from a CSV	`sales_df2 = pd.read_csv(“sales.csv”)`
Set the first column to the index, and parse dates	`sales_df2 = pd.read_csv(“sales.csv” index_col=0, parse_dates=True)`

pandas

Read the “games.csv” file	`games = pd.read_csv(“games.csv”)`
Take a look at the data	`games.head()`
View the columns, types and null value counts	`games.info()`
Get statistics about the numeric columns	`games.describe()`

Commercial Break: lambda

Anonymous function

lambda keyword

```
anon = lambda
```

parameter

```
anon = lambda x:
```

implicit return

```
anon = lambda x: x + 1
```

```
def inc(x):
  return x + 1
```

Works like any function

```
i = anon(2)
```

Or as a parameter

```
def inc(x, fn):
  return fn(x)

i = inc(2, lambda x: x + 4)
```

pandas

For today, the city names are superfluous. How can we remove them?

The apply method on a Series (or DataFrame) will accept a function (lambda) and pass every value to that function.

```
team_names = games[“winner”].apply(
  lambda team: team.split()[-1])
```

The result is a Series that can be assigned as a new column

```
games[“win_team_name”] = team_names
```

The winner column is now duplicative, remove it with the drop method.

```
games.drop(“winner”)
```

Remember to use the correct axis, and to set the inplace keyword argument.

```
games.drop(“winner”, axis=1, inplace=True)
```

pandas

Now that we have the winning team name, how do we find the losing team name?

If the winning team is the home team, the away team lost, and vice versa.

```
loss_team_name = row[“away_team_name”] if
  row[“home_team_name”] == row[“win_team_name”] else row[“home_team_name”]
```

Use this conditional in a lambda and pass it to the apply method, called on the DataFrame

```
games.apply(lambda row: ...)
```

Once more, set the axis and assign the result to a new column

```
games[“loss_team_name”] =
  games.apply(lambda row: ..., axis=1)
```

Drop the home_team, away_team, home_team_city, and away_team_city columns

```
games.drop([“home_team”, “away_team”, ...],
  axis=1, inplace=True)
```

pandas

Create a filter to include only Titans home games

```
titans_home_games =
  games[“home_team_name”] == “Titans”
```

Use the Series to return a new DataFrame

```
games[titans_home_games]
```

Now filter for the games the Titans won

```
titans_wins =
  games[“win_team_name”] == “Titans”
```

Use the and (&) and or (|) operators to combine the filter and return a DataFrame with Titans home wins

```
titans_home_wins = games[
  titans_home_games & titans_wins]
```

What was the Titans average winning score?

```
titans_home_wins[“pts_win”].mean()
```

What was the average spread for a Titans win?

```
titans_home_wins[“spread”] = titans_home_wins[“pts_win”] – titans_home_wins[“pts_loss”]
titans_home_wins[“spread”].mean()
```

pandas

Sort the Titans wins by the winning score

```
titans_home_wins.sort_values(“pts_win”)
```

Change the order from greatest to least

```
titans_home_wins.sort_values(“pts_win”,
  ascending=False)
```

Sort by winning score and then spread. Winning score in descending order, spread in ascending order.

```
titans_home_wins.sort_values([“pts_win”,
  “spread”, ascending=[False, True])
```

Sort the Titans games in the year 2000 by week

```
titans_home_games = games[“home_team_name”] == “Titans”
titans_away_games = games[“away_team_name”] == “Titans”
games_2000 = games[“year”] == 2000
titans_games_2000 = games[(titans_home_games | titans_away_games) & games_2000]
titans_games_2000.sort_values(“week”)
```

Take a look at the week column. Week 10 comes before week 2 and the Super Bowl before Wild Card.

pandas

Why is this? Look at the unique values in the week column

```
titans_games_2000[“week”].unique()
```

The values are all strings. So how can we tell pandas to order the weeks correctly?

Passing a string representation of an integer to the int initializer will return the integer value

```
int(“1”)
```

And the playoff weeks can be mapped to integer values using a dictionary

```
playoff_week_order = {
  “WildCard”: 18,
  “Division”: 19,
  “ConfChamp”: 20,
  “SuperBowl”: 21,
}
```

pandas

Write a new function, set_week_order that accepts a value from the week column

The function will attempt convert the week to an integer

For the regular season, this will work, but for the playoffs a ValueError will be raised as you can't cast "SuperBowl" to an int.

```
def set_week_order(week):
```

```
def set_week_order(week):
  return int(week)
```

Commercial Break: try-except

When running code that could raise an error, you place it in a try block:

```
try:
  print(int(“SuperBowl”))
```

If the code in the try block raises an error, flow will jump to an except block

```
except ValueError as e:
  print(“Something went wrong”)
```

For the set_week_order function, we can attempt to return the int value of the week. If it raises an error, try getting the int value from the playoffs_week_order dictionary.

```
try:
  return int(week)
except ValueError as e:
  return playoffs_week_order[week]
```

Using the apply method on the week column, pass the set_week_order function and assign the returned series to a new week_order column

```
games[“week_order”] = games[“week”].apply(
  set_week_order)
```

```
games_2019 = games[“year”] == 2019
titans_games_2019 = games[(titans_home_games | titans_away_games) & games_2019]
titans_games_2019.sort_values(“week_order”)
```

pandas

Get all Titans playoff games

```
playoff_weeks = games[“week_order”] >= playoffs_week_order[“WildCard”]
titans_games = titans_home_games | titans_away_games
titans_playoffs = games[titans_games & playoff_weeks]
titans_playoffs.sort_values([“year”, “week_order”])
```

Which team won the most Super Bowls? (2000-2019)

```
super_bowls = games[games[“week_order”] == playoffs_week_order[“SuperBowl”]]
super_bowls[“win_team_name”].value_counts()
```

What is the highest winning score in 2015?

```
games_2015 = games[“year”] == 2015
games[games_2015][“pts_win”].max()
```

Your turn!

Find the highest losing score during the regular season in the years 2004-2007.

Extra credit: During the 2004-2007 regular season the average losing score was about 17. Which team had that losing score the most?

pandas

You can group a DataFrame by a column

```
games.groupby([“year”])
```

Compute summary statistics on the groups

```
games.groupby([“year”])
    .sum(numeric_only=True)
```

Which team won the most Super Bowls? (2000-2019)

```
super_bowls = games[games[“week_order”] == playoffs_week_order[“SuperBowl”]]
super_bowls[“win_team_name”].value_counts()
```

What is the highest winning score in 2015?

```
games_2015 = games[“year”] == 2015
games[games_2015][“pts_win”].max()
```

pandas

Create two random DataFrames

```
left_array = rng.integers(1, 10, (4, 4,))
right_array = rng.integers(1, 10, (4, 4,))
```

```
import string
left_columns =
  list(string.ascii_letters[:4])
right_columns =
  list(string.ascii_letters[4:8])
```

```
left = pd.DataFrame(
  data=left_array, columns=left_columns)
right = pd.DataFrame(
  data=right_array, columns=right_columns)
```

Merge the left DataFrame with the right

```
left.merge(right)
```

Add a common column

```
common = fake.words(8)
```

```
left[“common”] = common[:4]
right[“common”] = common[:4]
```

Try the merge again

```
left.merge(right)
```

pandas

By default, pandas will attempt an inner merge. An inner merge will keep only rows that have the same key value in both DataFrames.

```
left.merge(right, how=“inner”)
```

Modify the common column of the right DataFrame

```
right[“common”] = common[1:5]
```

Attempt the inner merge again

```
left.merge(right)
```

A left merge will keep all rows in the left DataFrame. Rows without matching keys in the right DataFrame will be filled with NaN

```
left.merge(right, how=“left”)
```

A right merge keeps the rows in the right DataFrame

```
left.merge(right, how=“right”)
```

An outer merge keeps the rows in both DataFrames

```
left.merge(right, how=“outer”)
```

If the DataFrames have no common columns, you can specify the columns to merge with left_on and right_on keywords

```
right[“new_column”] = right[“common”]
del right[“new_column”]
left.merge(right, left_on=“common”,
  right_on=“new_column”)
```

Your turn!

Merge the data from the files “games_2000_w_1.csv” and “home_attendance_w_1.csv”

pandas

At no extra charge, pandas provides the join method which is a specialization of merge

```
left.join(right)
```

The join method uses indexes and left join by default.

The concat function in pandas will “stack” multiple DataFrames vertically by default

```
pd.concat([left, right])
```

Concat does an outer join by default

Matplotlib

decisions are worth dollars

matplotlib

Look at the weekly attendance of Titans games in 2019

matplotlib

Look at the weekly attendance of Titans games in 2019

```
import pandas as pd
```

matplotlib

Look at the weekly attendance of Titans games in 2019

```
import pandas as pd
attendance = pd.read_csv("attendance.csv")
```

matplotlib

Look at the weekly attendance of Titans games in 2019

```
import pandas as pd

attendance = pd.read_csv(“attendance.csv”)

titans_2019 = attendance[attendance[“team_name”] == “Titans” &
  attendance[“year”] == 2019]
```

matplotlib

Look at the weekly attendance of Titans games in 2019

```
import pandas as pd

attendance = pd.read_csv(“attendance.csv”)

titans_2019 = attendance[attendance[“team_name”] == “Titans” &
  attendance[“year”] == 2019]

titans_2019.dropna(inplace=True)
```

matplotlib

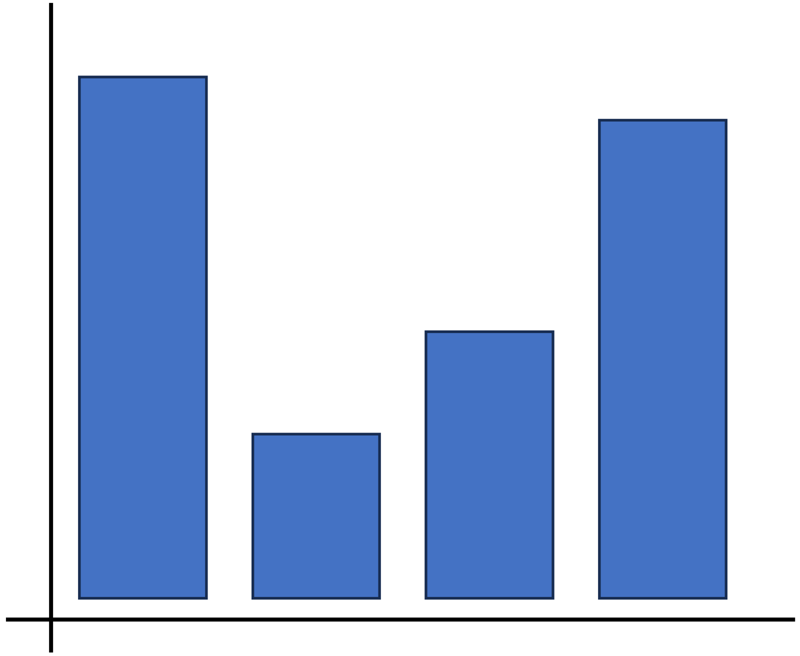
Look at the weekly attendance of Titans games in 2019

```
import pandas as pd

attendance = pd.read_csv(“attendance.csv”)

titans_2019 = attendance[attendance[“team_name”] == “Titans” &
  attendance[“year”] == 2019]

titans_2019.dropna(inplace=True)
```

Which week had the highest attendance?

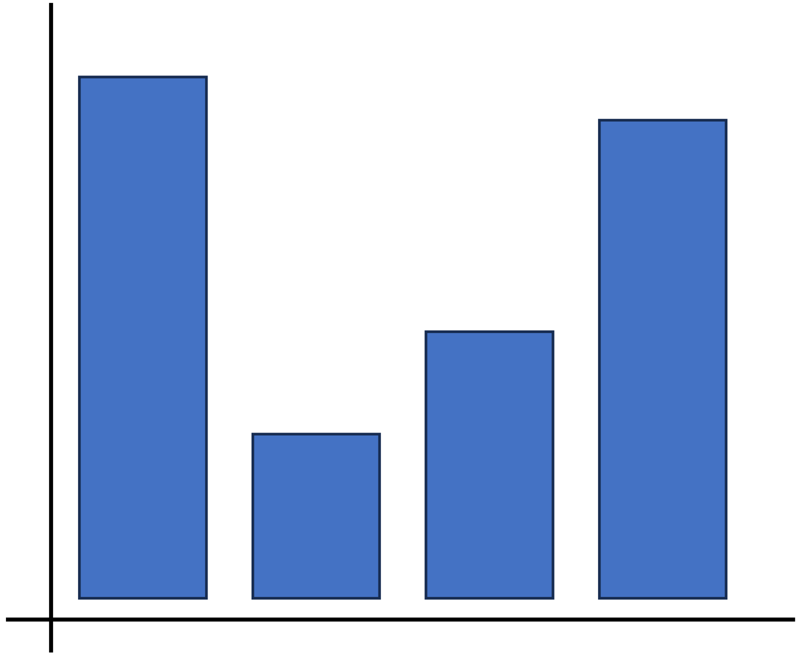
TIMES UP!

Bar Chart



Used to compare quantities

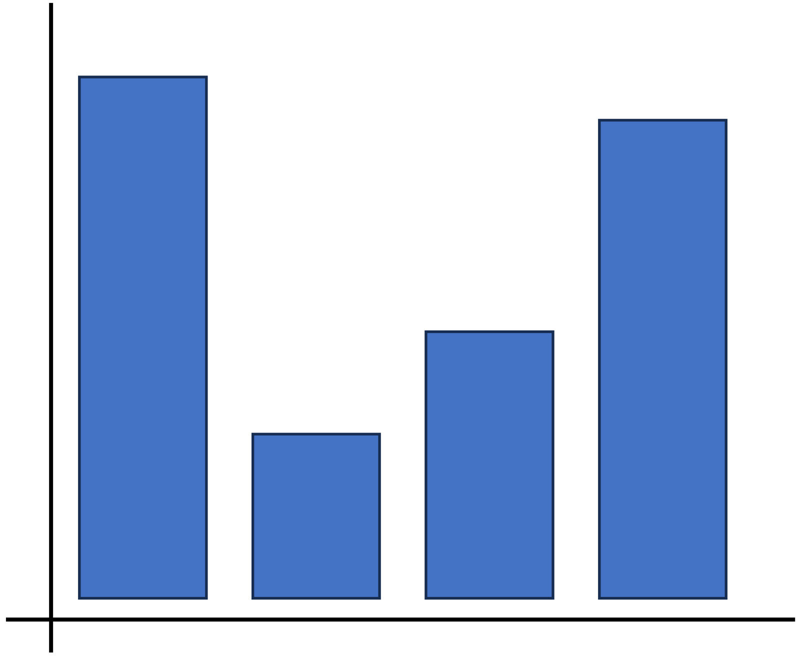
Bar Chart



Used to compare quantities

Also called a column chart

Bar Chart

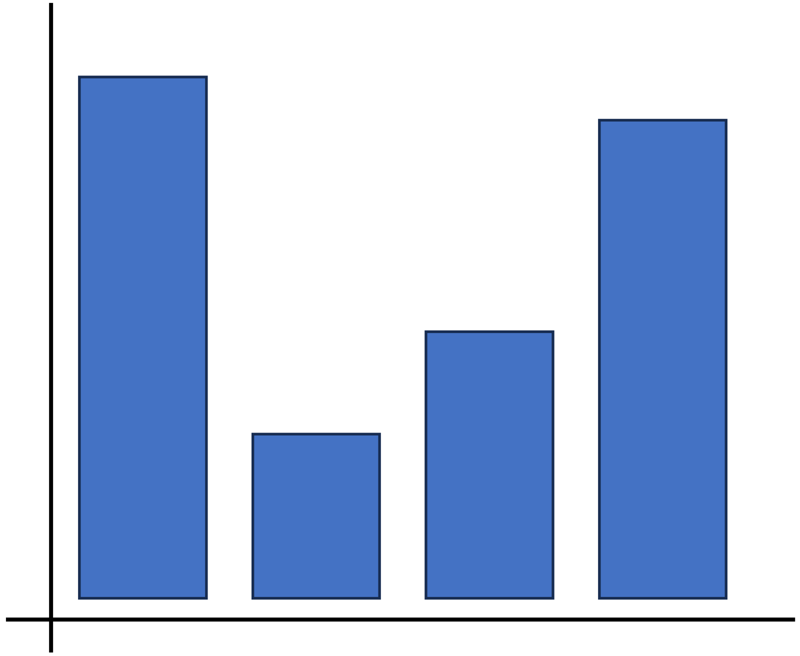


Used to compare quantities

Also called a column chart

Sometimes displayed horizontally

Bar Chart



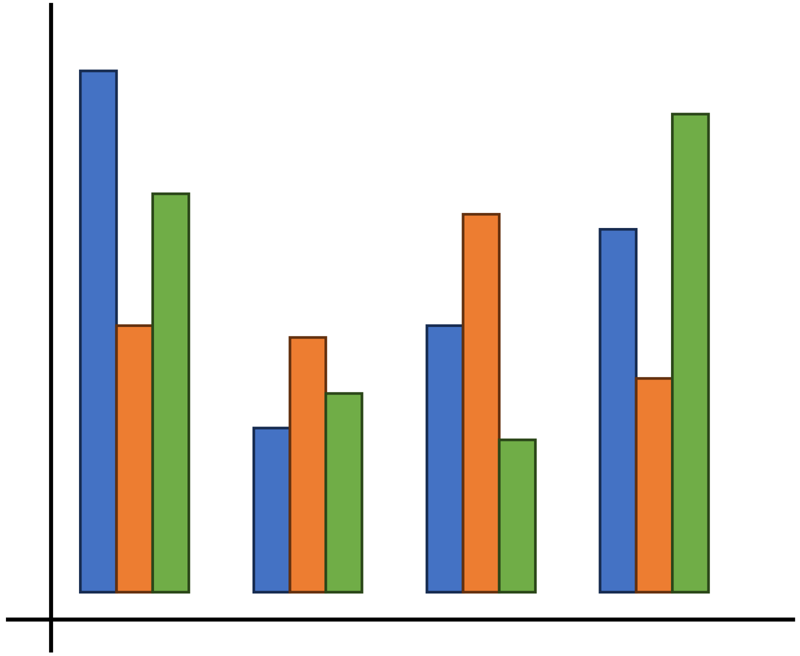
Used to compare quantities

Also called a column chart

Sometimes displayed horizontally

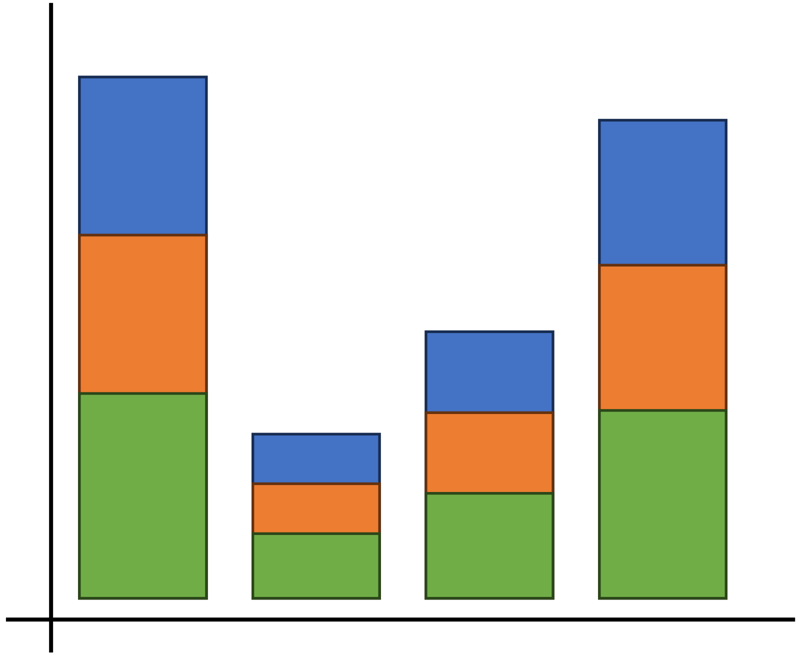
Good at showing time series

Clustered Bar Chart



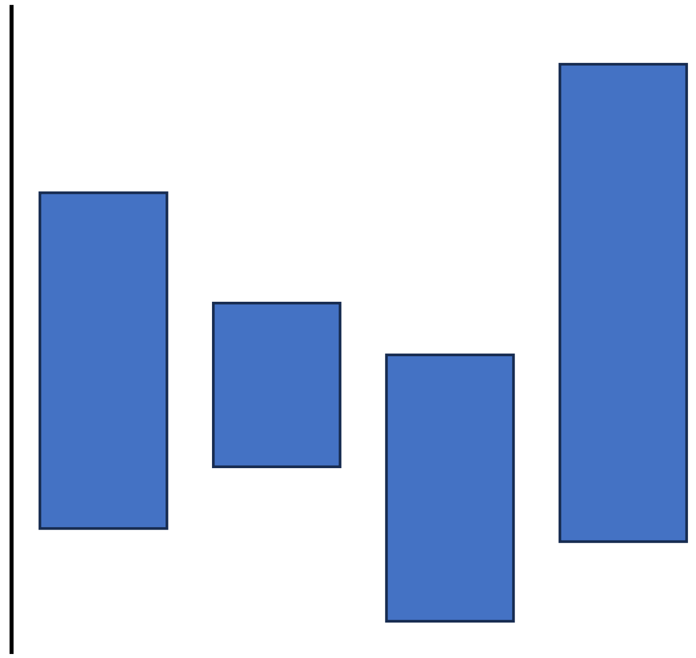
Used to compare multiple sets of data

Stacked Bar Chart



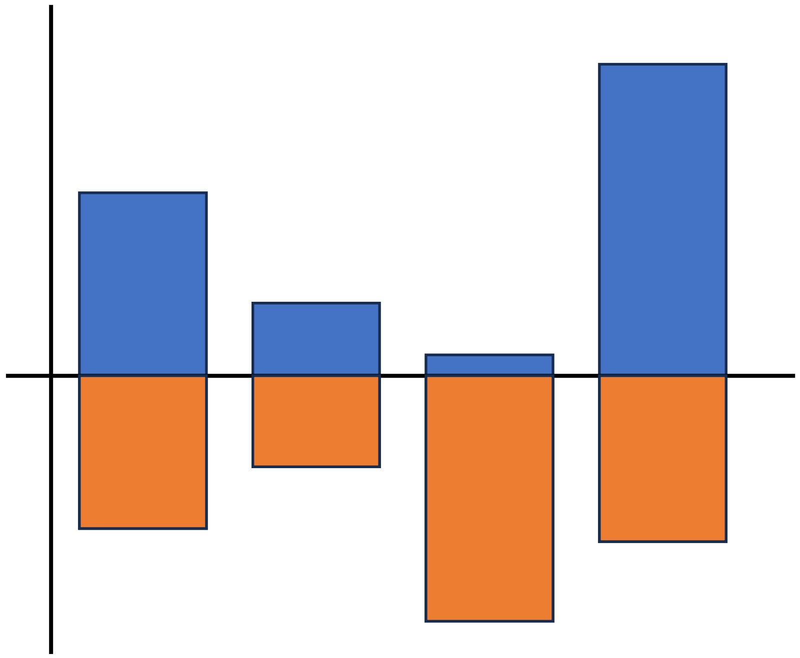
Used to compare values in a category

Floating Bar Chart



Not all bars start at the same value

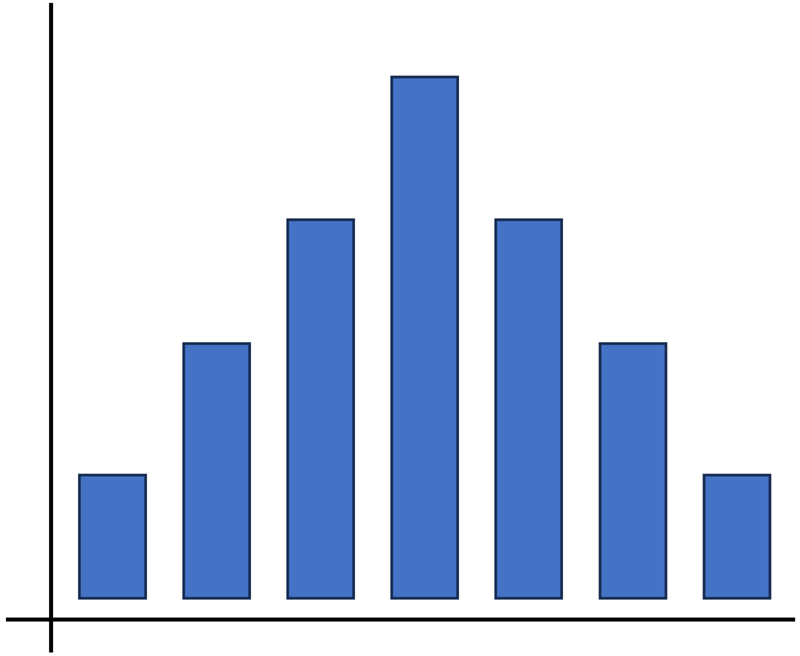
Floating Bar Chart



Not all bars start at the same value

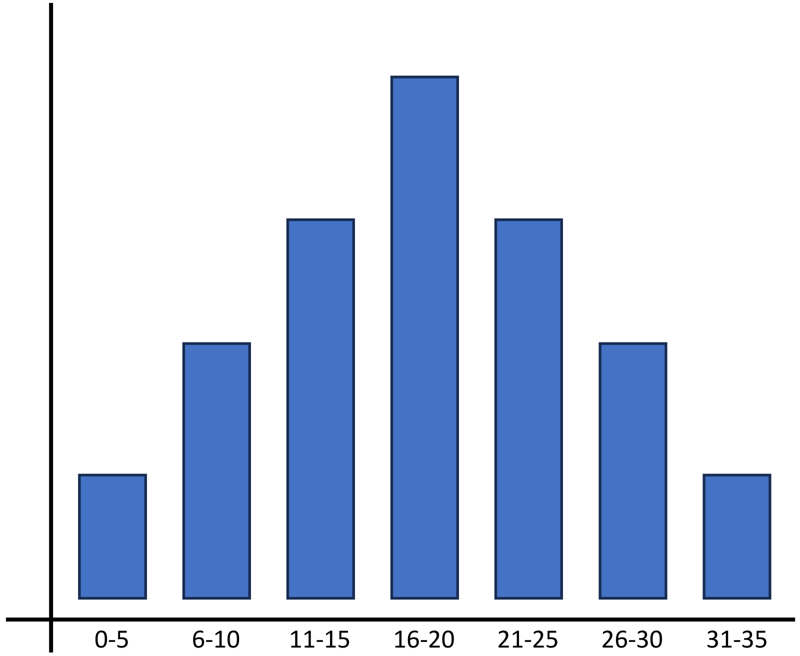
Shows positive and negative

Histogram



Show a distribution of values

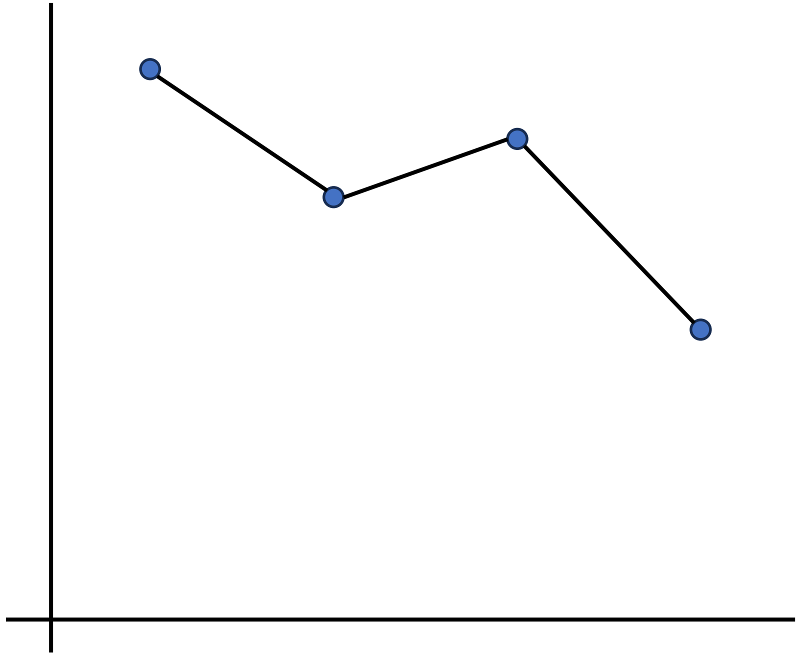
Histogram



Show a distribution of values

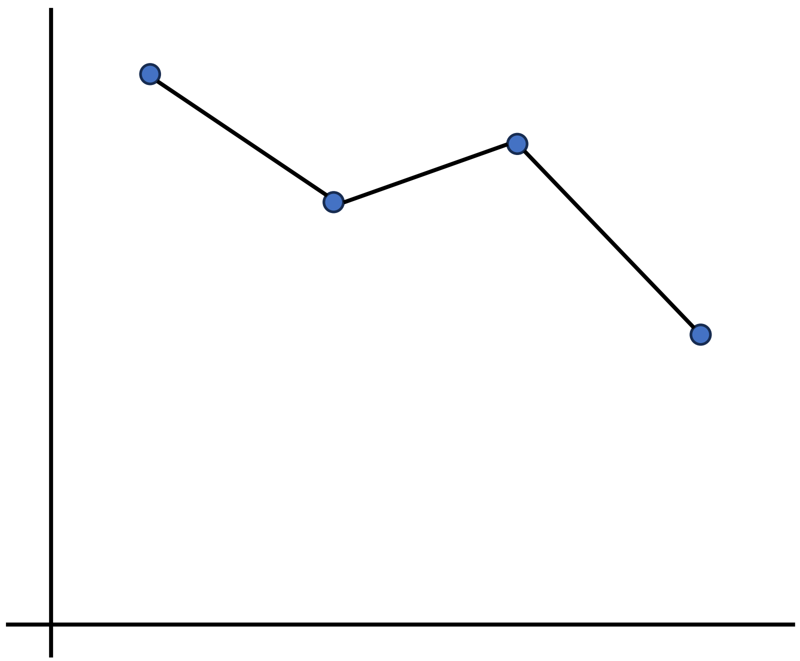
Each bar represents a count in a subset

Line Chart



Used to display continuous data

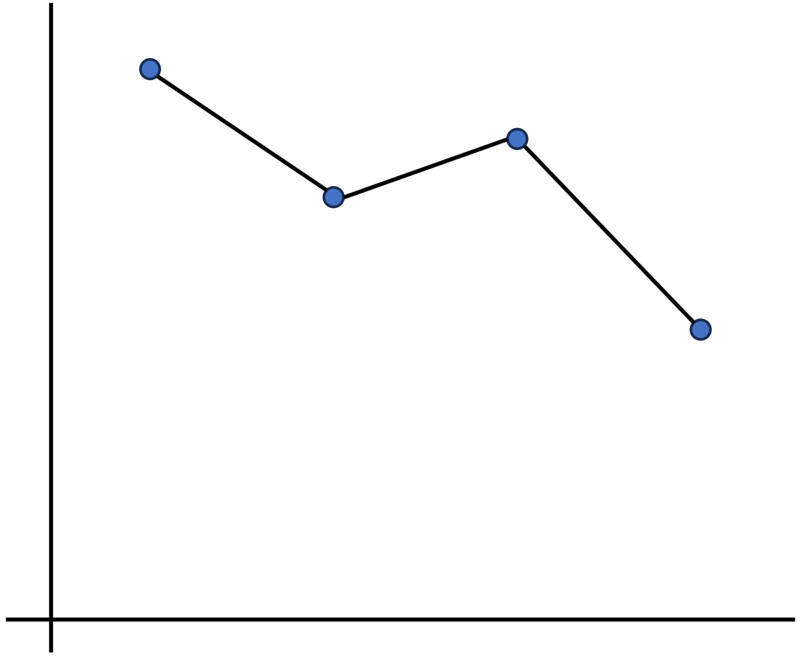
Line Chart



Used to display continuous data

Visualizes trends

Line Chart

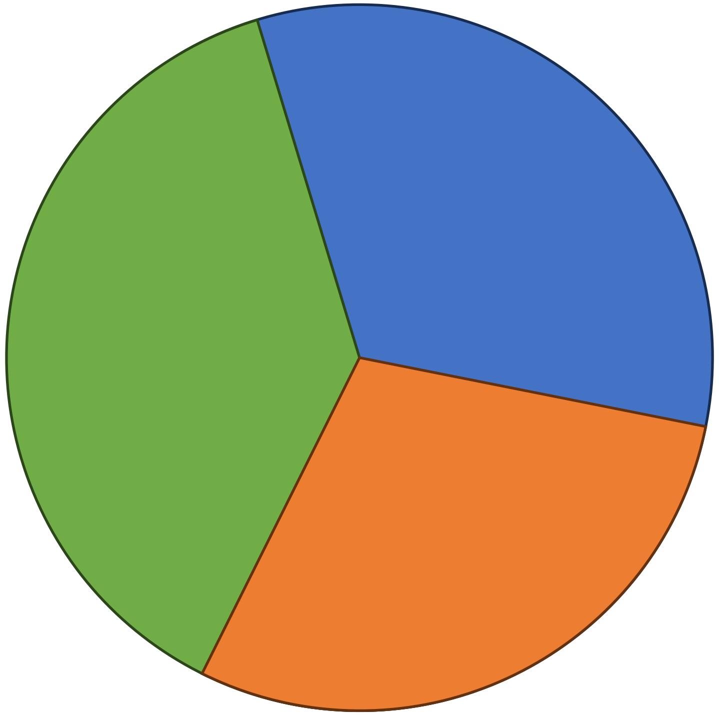


Used to display continuous data

Visualizes trends

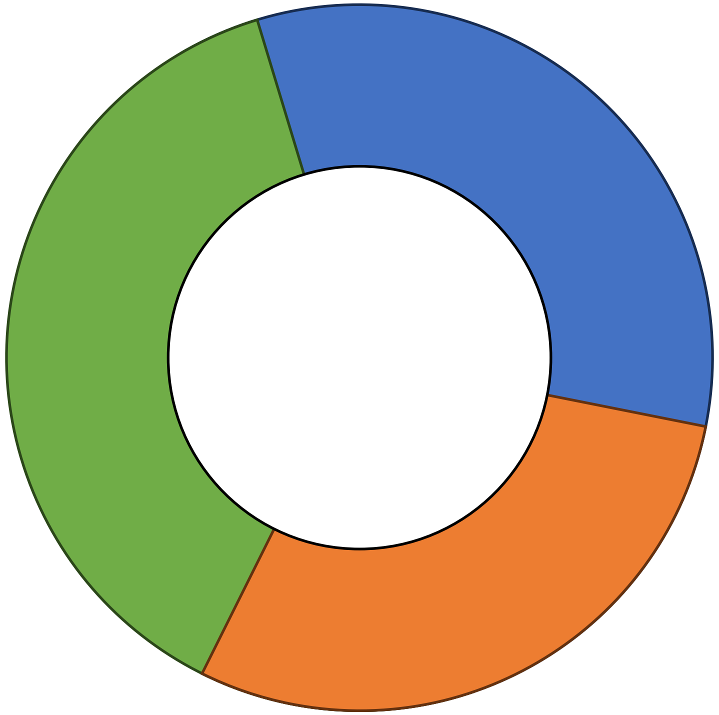
Relationships over time

Pie Chart



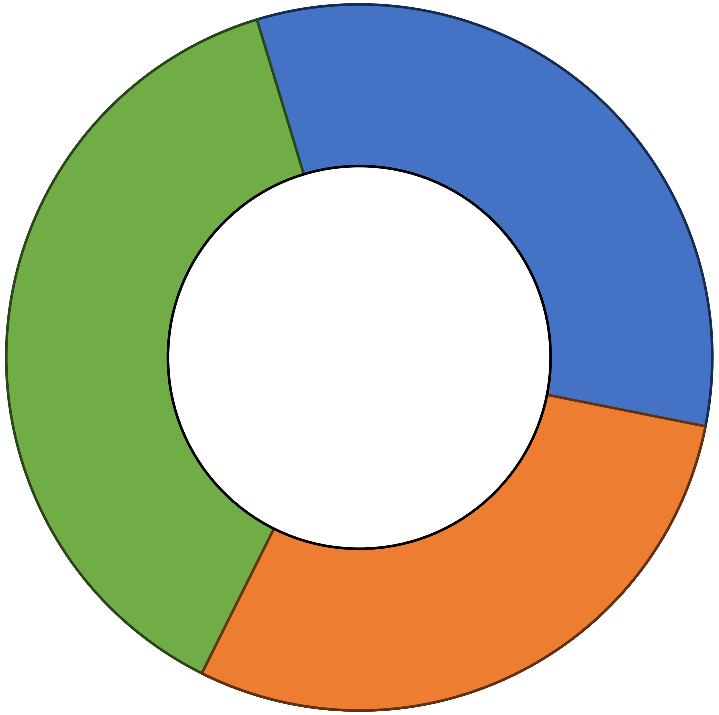
Used to show the parts of a whole

Donut Chart



Pie chart with space in the middle

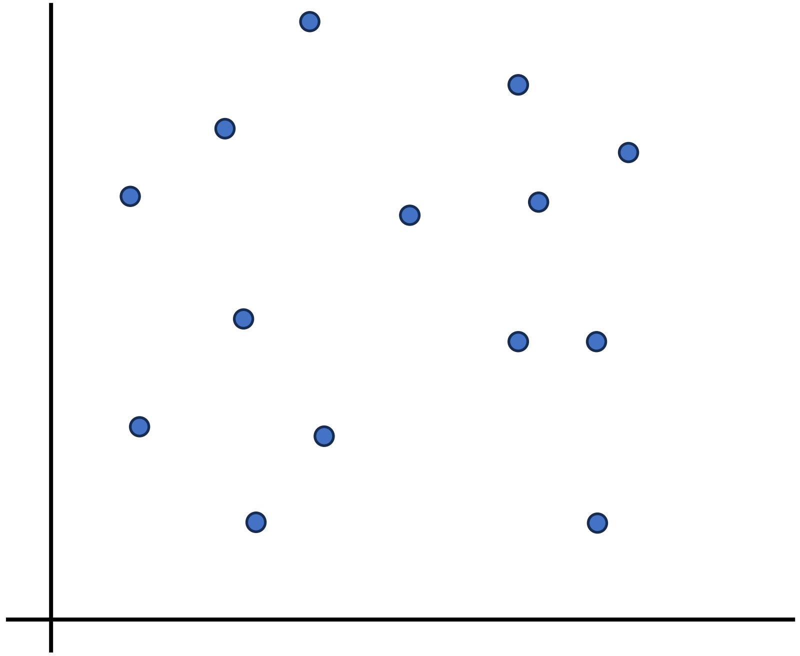
Donut Chart



Pie chart with space in the middle

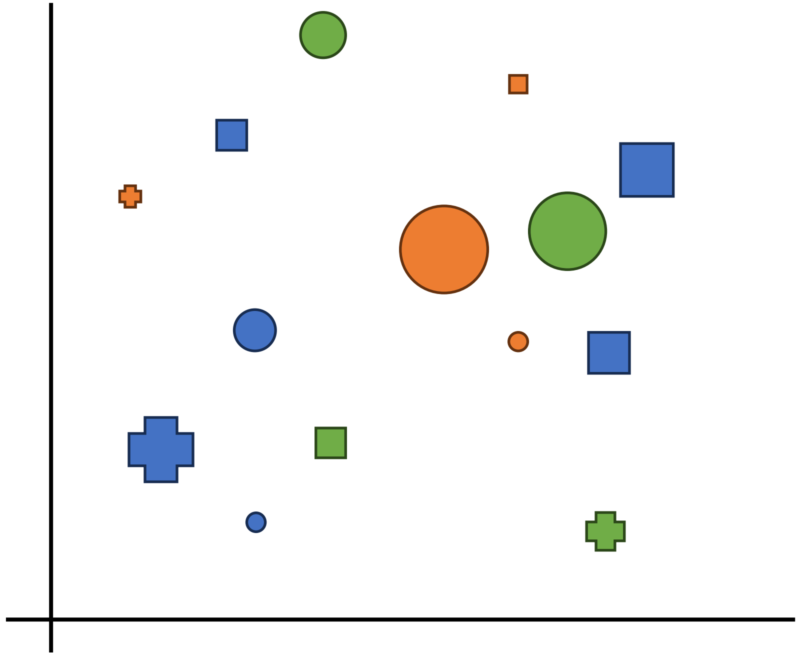
Use the space to show summary, details, a logo

Scatter Chart



Shows the relationship between two variables at the same time

Scatter Chart



Shows the relationship between two variables at the same time

Additional data can be shown using color, symbol and size

Matplotlib

The plot method will create a line chart.

The cumsum method will generate a cumulative sum at every row in a DataFrame

```
plt.plot(rng.integers(0, 10, (10,)))
```

```
def plot_team_cum_pts(team, year):
  home_games = games[“home_team_name”] == team
  away_games = games[“away_team_name”] == team
  year_games = games[“year”] == year
  df_year = games[(home_games | away_games) & year_games].copy()
  year_pts = df_year.apply(lambda row:
    row[“pts_win”] if row[“win_team_name”] == team else row[“pts_loss”], axis=1)
  cum_pts = year_pts.cumsum()
  plt.plot(cum_pts.values)
```

Matplotlib

Read the file clusters.csv into a DataFrame

Matplotlib

Read the file clusters.csv into a DataFrame

```
clusters = pd.read_csv(“clusters.csv”)
```

Matplotlib

Read the file clusters.csv into a DataFrame

Use the x and y columns to create a scatter plot

```
clusters = pd.read_csv(“clusters.csv”)
```

Matplotlib

Read the file clusters.csv into a DataFrame

```
clusters = pd.read_csv(“clusters.csv”)
```

Use the x and y columns to create a scatter plot

```
plt.scatter(clusters[“x”], clusters[“y”])
```

Matplotlib

Read the file clusters.csv into a DataFrame

Use the x and y columns to create a scatter plot

It looks like there are only two clusters. Use the "cluster" column to color the points in each one.

```
clusters = pd.read_csv("clusters.csv")

plt.scatter(clusters["x"], clusters["y"])
```

Matplotlib

Read the file clusters.csv into a DataFrame

```
clusters = pd.read_csv(“clusters.csv”)
```

Use the x and y columns to create a scatter plot

```
plt.scatter(clusters[“x”], clusters[“y”])
```

It looks like there are only two clusters. Use the “cluster” column to color the points in each one.

```
plt.scatter(clusters[“x”], clusters[“y”],
  c=clusters[“cluster”])
```

Matplotlib

Read the file clusters.csv into a DataFrame

```
clusters = pd.read_csv("clusters.csv")
```

Use the x and y columns to create a scatter plot

```
plt.scatter(clusters["x"], clusters["y"])
```

It looks like there are only two clusters. Use the "cluster" column to color the points in each one.

```
plt.scatter(clusters["x"], clusters["y"],
  c=clusters["cluster"])
```

Finally, use the size column to scale the points.

Matplotlib

Read the file clusters.csv into a DataFrame

```
clusters = pd.read_csv(“clusters.csv”)
```

Use the x and y columns to create a scatter plot

```
plt.scatter(clusters[“x”], clusters[“y”])
```

It looks like there are only two clusters. Use the “cluster” column to color the points in each one.

```
plt.scatter(clusters[“x”], clusters[“y”],
  c=clusters[“cluster”])
```

Finally, use the size column to scale the points.

```
plt.scatter(clusters[“x”], clusters[“y”],
  c=clusters[“cluster”], s=clusters[“size”] * 100)
```

Matplotlib

Create a NumPy array with a normal distribution

Matplotlib

Create a NumPy array with a normal distribution

```
normal = np.random.normal(size=1000)
```

Matplotlib

Create a NumPy array with a normal distribution

Create a histogram

```
normal = np.random.normal(size=1000)
```

Matplotlib

Create a NumPy array with a normal distribution	`normal = np.random.normal(size=1000)`
Create a histogram	`plt.hist(normal)`

Matplotlib

Create a NumPy array with a normal distribution

```
normal = np.random.normal(size=1000)
```

Create a histogram

```
plt.hist(normal)
```

By default a histogram has 10 bins. The bins keyword can set the number of bins.

Matplotlib

Create a NumPy array with a normal distribution	`normal = np.random.normal(size=1000)`
Create a histogram	`plt.hist(normal)`
By default a histogram has 10 bins. The bins keyword can set the number of bins.	`plt.hist(normal, bins=20)`

Matplotlib

Create a NumPy array with a normal distribution	`normal = np.random.normal(size=1000)`
Create a histogram	`plt.hist(normal)`
By default a histogram has 10 bins. The bins keyword can set the number of bins.	`plt.hist(normal, bins=20)`
Create a NumPy array with a uniform distribution	

Matplotlib

Create a NumPy array with a normal distribution	`normal = np.random.normal(size=1000)`
Create a histogram	`plt.hist(normal)`
By default a histogram has 10 bins. The bins keyword can set the number of bins.	`plt.hist(normal, bins=20)`
Create a NumPy array with a uniform distribution	`uniform = np.random.uniform(size=1000)`

Matplotlib

Create a NumPy array with a normal distribution	`normal = np.random.normal(size=1000)`
Create a histogram	`plt.hist(normal)`
By default a histogram has 10 bins. The bins keyword can set the number of bins.	`plt.hist(normal, bins=20)`
Create a NumPy array with a uniform distribution	`uniform = np.random.uniform(size=1000)`
Create a histogram	

Matplotlib

Create a NumPy array with a normal distribution	`normal = np.random.normal(size=1000)`
Create a histogram	`plt.hist(normal)`
By default a histogram has 10 bins. The bins keyword can set the number of bins.	`plt.hist(normal, bins=20)`
Create a NumPy array with a uniform distribution	`uniform = np.random.uniform(size=1000)`
Create a histogram	`plt.hist(uniform)`

Matplotlib

Create a NumPy array with a normal distribution

```
normal = np.random.normal(size=1000)
```

Create a histogram

```
plt.hist(normal)
```

By default a histogram has 10 bins. The bins keyword can set the number of bins.

```
plt.hist(normal, bins=20)
```

Create a NumPy array with a uniform distribution

```
uniform = np.random.uniform(size=1000)
```

Create a histogram

```
plt.hist(uniform)
```

Plot two normal distributions in a scatter chart.
What do you think it will look like?

Matplotlib

Create a NumPy array with a normal distribution	`normal = np.random.normal(size=1000)`
Create a histogram	`plt.hist(normal)`
By default a histogram has 10 bins. The bins keyword can set the number of bins.	`plt.hist(normal, bins=20)`
Create a NumPy array with a uniform distribution	`uniform = np.random.uniform(size=1000)`
Create a histogram	`plt.hist(uniform)`
Plot two normal distributions in a scatter chart. What do you think it will look like?	`plt.scatter(np.random.normal(size=1000), np.random.normal(size=1000))`

Matplotlib

Create a NumPy array with a normal distribution	`normal = np.random.normal(size=1000)`
Create a histogram	`plt.hist(normal)`
By default a histogram has 10 bins. The bins keyword can set the number of bins.	`plt.hist(normal, bins=20)`
Create a NumPy array with a uniform distribution	`uniform = np.random.uniform(size=1000)`
Create a histogram	`plt.hist(uniform)`
Plot two normal distributions in a scatter chart. What do you think it will look like?	`plt.scatter(np.random.normal(size=1000), np.random.normal(size=1000))`
How would using a uniform distribution differ?	

Matplotlib

Create a NumPy array with a normal distribution	`normal = np.random.normal(size=1000)`
Create a histogram	`plt.hist(normal)`
By default a histogram has 10 bins. The bins keyword can set the number of bins.	`plt.hist(normal, bins=20)`
Create a NumPy array with a uniform distribution	`uniform = np.random.uniform(size=1000)`
Create a histogram	`plt.hist(uniform)`
Plot two normal distributions in a scatter chart. What do you think it will look like?	`plt.scatter(np.random.normal(size=1000), np.random.normal(size=1000))`
How would using a uniform distribution differ?	`plt.scatter(np.random.uniform(size=1000), np.random.uniform(size=1000))`

Matplotlib

Matplotlib is a Python package for visualizing data

Matplotlib

Matplotlib is a Python package for visualizing data

Import the matplotlib.pyplot module

Matplotlib

Matplotlib is a Python package for visualizing data

Import the matplotlib.pyplot module

```
import matplotlib.pyplot
```

Matplotlib

Matplotlib is a Python package for visualizing data

Import the matplotlib.pyplot module (this is where aliases are really nice!)

```
import matplotlib.pyplot as plt
```

Matplotlib

Matplotlib is a Python package for visualizing data

Import the matplotlib.pyplot module (this is where aliases are really nice!)

```
import matplotlib.pyplot as plt
```

The bar function will create a bar chart

Matplotlib

Matplotlib is a Python package for visualizing data

Import the matplotlib.pyplot module (this is where aliases are really nice!)

```
import matplotlib.pyplot as plt
```

The bar function will create a bar chart

```
plt.bar()
```

Matplotlib

Matplotlib is a Python package for visualizing data

Import the matplotlib.pyplot module (this is where aliases are really nice!)

```
import matplotlib.pyplot as plt
```

The bar function will create a bar chart

```
plt.bar()
```

It expects the values for the x-axis

Matplotlib

Matplotlib is a Python package for visualizing data

Import the matplotlib.pyplot module (this is where aliases are really nice!)

```
import matplotlib.pyplot as plt
```

The bar function will create a bar chart

```
plt.bar()
```

It expects the values for the x-axis

```
plt.bar(
  np.arange(len(titans_2019)),
)
```

Matplotlib

Matplotlib is a Python package for visualizing data

Import the matplotlib.pyplot module (this is where aliases are really nice!)

```
import matplotlib.pyplot as plt
```

The bar function will create a bar chart

```
plt.bar()
```

It expects the values for the x-axis

```
plt.bar(
  np.arange(len(titans_2019)),
)
```

And values for the y-axis

Matplotlib

Matplotlib is a Python package for visualizing data

Import the matplotlib.pyplot module (this is where aliases are really nice!)

```
import matplotlib.pyplot as plt
```

The bar function will create a bar chart

```
plt.bar()
```

It expects the values for the x-axis

```
plt.bar(
  np.arange(len(titans_2019)),
)
```

And values for the y-axis

```
plt.bar(
  np.arange(len(titans_2019)),
  titans_2019[“weekly_attendance”])
```

Matplotlib

Matplotlib is a Python package for visualizing data

Import the matplotlib.pyplot module (this is where aliases are really nice!)

```
import matplotlib.pyplot as plt
```

The bar function will create a bar chart

```
plt.bar()
```

It expects the values for the x-axis

```
plt.bar(
  np.arange(len(titans_2019)),
)
```

And values for the y-axis

```
plt.bar(
  np.arange(len(titans_2019)),
  titans_2019[“weekly_attendance”])
```

Already, it’s easy at a glance to see the highest (and lowest) values in the data

Matplotlib

The xticks method formats the ticks on the x-axis

Matplotlib

The xticks method formats the ticks on the x-axis

```
plt.xticks()
```

Matplotlib

The xticks method formats the ticks on the x-axis

It accepts the positions of the ticks

```
plt.xticks()
```

Matplotlib

The xticks method formats the ticks on the x-axis

```
plt.xticks()
```

It accepts the positions of the ticks

```
plt.xticks(np.arange(len(titans_2019)))
```

Matplotlib

The xticks method formats the ticks on the x-axis

```
plt.xticks()
```

It accepts the positions of the ticks

```
plt.xticks(np.arange(len(titans_2019)))
```

And the labels for the ticks

Matplotlib

The xticks method formats the ticks on the x-axis

```
plt.xticks()
```

It accepts the positions of the ticks

```
plt.xticks(np.arange(len(titans_2019)))
```

And the labels for the ticks

```
plt.xticks(
  np.arange(len(titans_2019)),
  titans_2019[“week”])
```

Matplotlib

The xticks method formats the ticks on the x-axis

```
plt.xticks()
```

It accepts the positions of the ticks

```
plt.xticks(np.arange(len(titans_2019)))
```

And the labels for the ticks

```
plt.xticks(
  np.arange(len(titans_2019)),
  titans_2019[“week”])
```

You can also add labels to the x and y axes

Matplotlib

The xticks method formats the ticks on the x-axis

```
plt.xticks()
```

It accepts the positions of the ticks

```
plt.xticks(np.arange(len(titans_2019)))
```

And the labels for the ticks

```
plt.xticks(
  np.arange(len(titans_2019)),
  titans_2019[“week”])
```

You can also add labels to the x and y axes

```
plt.xlabel(“Week”)
plt.ylabel(“Attendance”)
```

Matplotlib

The xticks method formats the ticks on the x-axis

```
plt.xticks()
```

It accepts the positions of the ticks

```
plt.xticks(np.arange(len(titans_2019)))
```

And the labels for the ticks

```
plt.xticks(
  np.arange(len(titans_2019)),
  titans_2019[“week”])
```

You can also add labels to the x and y axes

```
plt.xlabel(“Week”)
plt.ylabel(“Attendance”)
```

And a legend

Matplotlib

The xticks method formats the ticks on the x-axis

```
plt.xticks()
```

It accepts the positions of the ticks

```
plt.xticks(np.arange(len(titans_2019)))
```

And the labels for the ticks

```
plt.xticks(
  np.arange(len(titans_2019)),
  titans_2019[“week”])
```

You can also add labels to the x and y axes

```
plt.xlabel(“Week”)
plt.ylabel(“Attendance”)
```

And a legend

```
plt.legend([“Weekly Attendance”])
```

Matplotlib

The xticks method formats the ticks on the x-axis

```
plt.xticks()
```

It accepts the positions of the ticks

```
plt.xticks(np.arange(len(titans_2019)))
```

And the labels for the ticks

```
plt.xticks(
  np.arange(len(titans_2019)),
  titans_2019[“week”])
```

You can also add labels to the x and y axes

```
plt.xlabel(“Week”)
plt.ylabel(“Attendance”)
```

And a legend

```
plt.legend([“Weekly Attendance”])
```

Finally, a title at the top of the visualization

Matplotlib

The xticks method formats the ticks on the x-axis

```
plt.xticks()
```

It accepts the positions of the ticks

```
plt.xticks(np.arange(len(titans_2019)))
```

And the labels for the ticks

```
plt.xticks(
  np.arange(len(titans_2019)),
  titans_2019[“week”])
```

You can also add labels to the x and y axes

```
plt.xlabel(“Week”)
plt.ylabel(“Attendance”)
```

And a legend

```
plt.legend([“Weekly Attendance”])
```

Finally, a title at the top of the visualization

```
plt.title(“Weekly Attendance Titans 2019”)
```

Matplotlib

The xticks method formats the ticks on the x-axis

```
plt.xticks()
```

It accepts the positions of the ticks

```
plt.xticks(np.arange(len(titans_2019)))
```

And the labels for the ticks

```
plt.xticks(
  np.arange(len(titans_2019)),
  titans_2019[“week”])
```

You can also add labels to the x and y axes

```
plt.xlabel(“Week”)
plt.ylabel(“Attendance”)
```

And a legend

```
plt.legend([“Weekly Attendance”])
```

Finally, a title at the top of the visualization

```
plt.title(“Weekly Attendance Titans 2019”)
```

Now which week had the highest attendance?

Matplotlib

Create a new DataFrame for the year 2000

Add another bar chart for the 2000 attendance

```
plt.bar(
  np.arange(len(titans_2000)),
  titans_2000[“weekly_attendance”])
```

The 2000 bars are placed on top of the 2019 bars

To solve this, narrow the bars and offset them.

```
plt.bar(
  np.arange(len(titans_2000)) – 0.2,
  titans_2000[“weekly_attendance”], width=0.4)
plt.bar(
  np.arange(len(titans_2000)) + 0.2,
  titans_2019[“weekly_attendance”], width=0.4)
```

And update the legend to display both years

```
plt.legend([“2000”, “2019”])
```

Matplotlib

But now the weeks are incorrect for 2000!

Update the bars and xticks to display 17 values

The 2000 bars are placed on top of the 2019 bars

At the top of the cell, get the figure and axes. The figure is the entire visualization. The axes is the area with the chart. For this example, they appear to be the same thing.

```
fig, ax = plt.subplots()
```

Get the xticklabels.

```
ax.get_xticklabels()
```

The xtick labels are an iterable. Get the label for each bye week and set it to use bold weight.

```
ax.get_xticklabels()[2].set_weight("bold")
ax.get_xticklabels()[10].set_weight("bold")
```

Matplotlib

Create a DataFrame for 2009 and add a new bar. You'll need to change to width to 0.3. Offset the bars for 2000 and 2019 by 0.3. And update the legend.

Format the xtick label for the 2009 bye week as bold

It's starting to get confusing as to which season each bye week is in. Color each bye week to show the season it is in. First, set the bars to the Titans colors.

```
plt.bar(..., color="#4B92DB")
plt.bar(..., color="#C60C30")
plt.bar(..., color="#A5ACAF")
```

Now set the tick label colors for each year.

```
ax.get_xticklabels()[2].set_color("#4B92DB")
ax.get_xticklabels()[6].set_color("#C60C30")
ax.get_xticklabels()[10].set_color("#A5ACAF")
```

Update the xlabel

```
plt.xlabel("Week (bye weeks in bold)")
```

Matplotlib

Get the home and away attendance for all teams in the year 2000.

```
home_away_2000 = attendance[
  (attendance[“year”] == 2000 &
  (attendance[“week”] == 1)]
```

Get the home and away attendance for the Titans

```
titans_2000 = home_away_2000[
  home_away_2000[“team_name”] == “Titans”]
```

The pie function expects an iterable to be used for the slices.

```
plt.pie([titans_2000[“home”], titans_2000[“away”]])
```

The error says that x must be 1D. Right now titans_2000 is a DataFrame. It has only one row, but it’s still has more that one dimension.

Use iloc to get the row from titans_2000

Add labels and percentages to the pie graph.

```
plt.pie(..., labels=[“Home”, “Away”],
  autopct=“%.2f”)
```

Matplotlib

Get Titans attendance for 2019

```
titans_2019 = attendance[(attendance[“year”] == 2019) & (attendance[“team_name”] == “Titans”)]
titans_2019_weekly = titans_2019[[“week”, “weekly_attendance”]]
top_5 = titans_2019_weekly.sort_values(“weekly_attendance”, ascending=False)[:5]
top_1 = top_5[“weekly_attendance”].argmax()
dynamite = np.zeros((len(big_5,)))
dynamite[top_1] = 0.25
plt.pie(
  top_5[“weekly_attendance”],
  labels=[f”Week {week}” for week in top_5[“week”]],
  autopct=“%.2f”,
  explode=dynamite)
```

Matplotlib

Keep week and weekly_attendance columns

```
titans_2019 = attendance[(attendance["year"] == 2019) & (attendance["team_name"] == "Titans")]
titans_2019_weekly = titans_2019[["week", "weekly_attendance"]]
top_5 = titans_2019_weekly.sort_values("weekly_attendance", ascending=False)[:5]
top_1 = top_5["weekly_attendance"].argmax()
dynamite = np.zeros((len(big_5,)))
dynamite[top_1] = 0.25
plt.pie(
  top_5["weekly_attendance"],
  labels=[f"Week {week}" for week in top_5["week"]],
  autopct="%.2f",
  explode=dynamite)
```

Matplotlib

Sort by weekly_attendance in descending order and slice the first 5

```
titans_2019 = attendance[(attendance[“year”] == 2019) & (attendance[“team_name”] == “Titans”)]
titans_2019_weekly = titans_2019[[“week”, “weekly_attendance”]]
top_5 = titans_2019_weekly.sort_values(“weekly_attendance”, ascending=False)[:5]
top_1 = top_5[“weekly_attendance”].argmax()
dynamite = np.zeros((len(big_5,)))
dynamite[top_1] = 0.25
plt.pie(
  top_5[“weekly_attendance”],
  labels=[f”Week {week}” for week in top_5[“week”]],
  autopct=“%.2f”,
  explode=dynamite)
```

Matplotlib

Get the index of the maximum value

```
titans_2019 = attendance[(attendance["year"] == 2019) & (attendance["team_name"] == "Titans")]
titans_2019_weekly = titans_2019[["week", "weekly_attendance"]]
top_5 = titans_2019_weekly.sort_values("weekly_attendance", ascending=False)[:5]
top_1 = top_5["weekly_attendance"].argmax()
dynamite = np.zeros((len(big_5,)))
dynamite[top_1] = 0.25
plt.pie(
  top_5["weekly_attendance"],
  labels=[f"Week {week}" for week in top_5["week"]],
  autopct="%.2f",
  explode=dynamite)
```

Matplotlib

For the explode array, start with zeros the same length as the data

```
titans_2019 = attendance[(attendance[“year”] == 2019) & (attendance[“team_name”] == “Titans”)]
titans_2019_weekly = titans_2019[[“week”, “weekly_attendance”]]
top_5 = titans_2019_weekly.sort_values(“weekly_attendance”, ascending=False)[:5]
top_1 = top_5[“weekly_attendance”].argmax()
dynamite = np.zeros((len(big_5,)))
dynamite[top_1] = 0.25
plt.pie(
  top_5[“weekly_attendance”],
  labels=[f”Week {week}” for week in top_5[“week”]],
  autopct=“%.2f”,
  explode=dynamite)
```

Matplotlib

Replace the maximum index with an explode factor (0.0-1.0)

```
titans_2019 = attendance[(attendance[“year”] == 2019) & (attendance[“team_name”] == “Titans”)]
titans_2019_weekly = titans_2019[[“week”, “weekly_attendance”]]
top_5 = titans_2019_weekly.sort_values(“weekly_attendance”, ascending=False)[:5]
top_1 = top_5[“weekly_attendance”].argmax()
dynamite = np.zeros((len(big_5,)))
dynamite[top_1] = 0.25
plt.pie(
  top_5[“weekly_attendance”],
  labels=[f”Week {week}” for week in top_5[“week”]],
  autopct=“%.2f”,
  explode=dynamite)
```

Matplotlib

Use the explode keyword to offset the largest value

```
titans_2019 = attendance[(attendance[“year”] == 2019) & (attendance[“team_name”] == “Titans”)]
titans_2019_weekly = titans_2019[[“week”, “weekly_attendance”]]
top_5 = titans_2019_weekly.sort_values(“weekly_attendance”, ascending=False)[:5]
top_1 = top_5[“weekly_attendance”].argmax()
dynamite = np.zeros((len(big_5,)))
dynamite[top_1] = 0.25
plt.pie(
  top_5[“weekly_attendance”],
  labels=[f”Week {week}” for week in top_5[“week”]],
  autopct=“%.2f”,
  explode=dynamite)
```

Matplotlib

Create a visualization with 4 subplots

```
fig, ax = plt.subplots(2, 2)
```

The figure is the entire visualization. The axes are the subplots.

The axes are stored in a NumPy array.

```
ax[0, 0]
```

Charts are plotted on the axes

```
ax[0, 0].plot(np.sin(np.linspace(
  0, np.pi * 2, 361)))
ax[0, 1].plot(np.cos(np.linspace(
  0, np.pi * 2, 361)))
ax[1, 0].plot(np.sin(np.linspace(
  -np.pi, np.pi, 361)))
ax[1, 1].plot(np.cos(np.linspace(
  -np.pi, np.pi, 361)))
```

Machine Learning

A branch of artificial intelligence you will find it is very artificial and not that intelligent.

Artificial Intelligence

Artificial Intelligence

What is the goal of artificial intelligence?

Artificial Intelligence

2 + 2

Artificial Intelligence

2 + 2

4

Artificial Intelligence

2 + 2

((2 + 4 - 1) * 6 + 2) / 8

4

Artificial Intelligence

$2 + 2$	$((2 + 4 - 1) * 6 + 2) / 8$
4	4

Artificial Intelligence

2 + 2	((2 + 4 - 1) * 6 + 2) / 8
Human Process	
4	4

Artificial Intelligence

2 + 2	((2 + 4 - 1) * 6 + 2) / 8

Human Process

4	4

Human Response

Artificial Intelligence

2 + 2	((2 + 4 - 1) * 6 + 2) / 8
Human Process	AI Process
4	4
Human Response	

Artificial Intelligence

Human Process	AI Process
2 + 2	((2 + 4 - 1) * 6 + 2) / 8

Human Response	AI Response
4	4

Artificial Intelligence

What is the goal of artificial intelligence?

The goal of artificial intelligence is to **mimic human response**.

Artificial Intelligence

2 + 2

Human Process

((2 + 4 - 1) * 6 + 2) / 8

AI Process

4

Human Response

4

AI Response

Artificial Intelligence

2 + 2

Human Process

((2 + 4 - 1) * 6 + 2) / 8

AI Process

4

Human Response

4

AI Response

Artificial Intelligence

What is the goal of artificial intelligence?

The goal of artificial intelligence is to
mimic human response.

The end justifies the means

Machine Learning

What is the goal of machine learning?

Machine Learning

What is the goal of machine learning?

The goal of machine learning is to
mimic human response ...

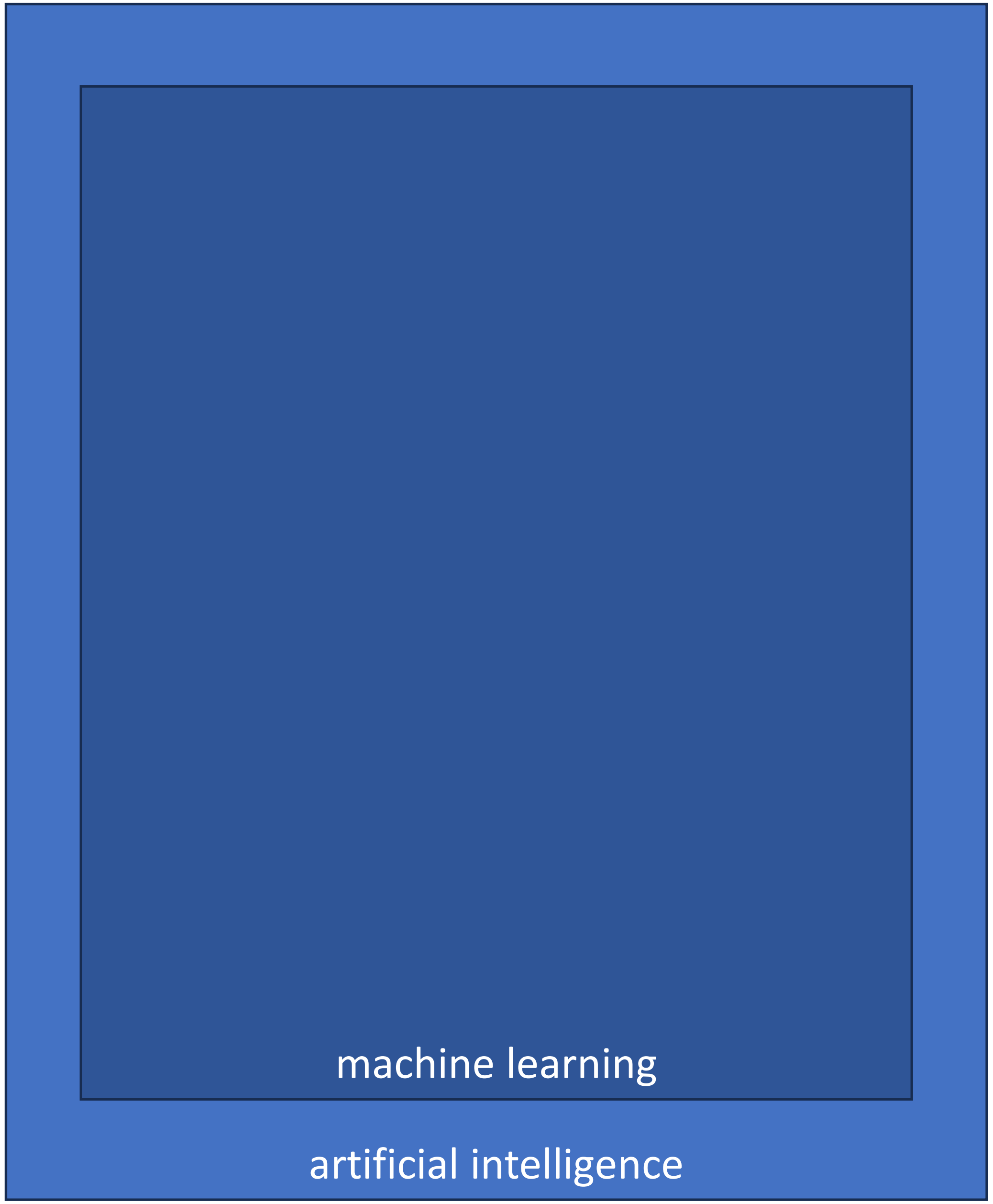
Machine Learning

What is the goal of machine learning?

The goal of machine learning is to
mimic human response ...

without being explicitly programmed.

artificial intelligence


machine learning
artificial intelligence

Machine Learning

What is the goal of machine learning?

The goal of machine learning is to
mimic human response ...

without being explicitly programmed.

Machine learning is a specialization of artificial intelligence.

Sepal Length	Sepal Width	Petal Length	Petal Width	Species
5.1	3.5	1.4	0.2	Setosa
4.9	3.0	1.4	0.2	Setosa
7.0	3.2	4.7	1.4	Versicolor
6.5	3.0	5.8	2.2	Virginica

Meet the iris dataset

Sepal Length	Sepal Width	Petal Length	Petal Width	Species
5.1	3.5	1.4	0.2	Setosa
4.9	3.0	1.4	0.2	Setosa
7.0	3.2	4.7	1.4	Versicolor
6.5	3.0	5.8	2.2	Virginica

Features are the values used to make a prediction.

Sepal Length	Sepal Width	Petal Length	Petal Width	Species
5.1	3.5	1.4	0.2	Setosa
4.9	3.0	1.4	0.2	Setosa
7.0	3.2	4.7	1.4	Versicolor
6.5	3.0	5.8	2.2	Virginica

The target is the value we want to predict

Sepal Length	Sepal Width	Petal Length	Petal Width	Species
5.1	3.5	1.4	0.2	Setosa
4.9	3.0	1.4	0.2	Setosa
7.0	3.2	4.7	1.4	Versicolor
6.5	3.0	5.8	2.2	Virginica

The iris dataset has three targets: setosa, versicolor and virginica

scikit-learn

Popular datasets (like iris) are included with scikit_learn

scikit-learn

Popular datasets (like iris) are included with scikit_learn

```
from sklearn.datasets import load_iris
iris = load_iris()
```

scikit-learn

Popular datasets (like iris) are included with scikit_learn

The features are in the data field

```
from sklearn.datasets import load_iris
iris = load_iris()
```

scikit-learn

Popular datasets (like iris) are included with scikit_learn

```
from sklearn.datasets import load_iris
iris = load_iris()
```

The features are in the data field

```
features = iris.data
```

scikit-learn

Popular datasets (like iris) are included with scikit_learn

```
from sklearn.datasets import load_iris
iris = load_iris()
```

The features are in the data field

```
features = iris.data
```

The targets are in the target field

scikit-learn

Popular datasets (like iris) are included with scikit_learn

```
from sklearn.datasets import load_iris
iris = load_iris()
```

The features are in the data field

```
features = iris.data
```

The targets are in the target field

```
targets = iris.target
```

scikit-learn

Popular datasets (like iris) are included with scikit_learn

```
from sklearn.datasets import load_iris
iris = load_iris()
```

The features are in the data field

```
features = iris.data
```

The targets are in the target field

```
targets = iris.target
```

A selection of the data will be reserved for testing the model after it is trained. It is important to ensure that the targets are equally represented in both the training and test datasets.

scikit-learn

Popular datasets (like iris) are included with scikit_learn

```
from sklearn.datasets import load_iris
iris = load_iris()
```

The features are in the data field

```
features = iris.data
```

The targets are in the target field

```
targets = iris.target
```

A selection of the data will be reserved for testing the model after it is trained. It is important to ensure that the targets are equally represented in both the training and test datasets.

The train_test_split function helps with this.

scikit-learn

Popular datasets (like iris) are included with scikit_learn

```
from sklearn.datasets import load_iris
iris = load_iris()
```

The features are in the data field

```
features = iris.data
```

The targets are in the target field

```
targets = iris.target
```

A selection of the data will be reserved for testing the model after it is trained. It is important to ensure that the targets are equally represented in both the training and test datasets.

The train_test_split function helps with this.

```
from sklearn.model_selection import
  train_test_split
```

scikit-learn

Popular datasets (like iris) are included with scikit_learn

```
from sklearn.datasets import load_iris
iris = load_iris()
```

The features are in the data field

```
features = iris.data
```

The targets are in the target field

```
targets = iris.target
```

A selection of the data will be reserved for testing the model after it is trained. It is important to ensure that the targets are equally represented in both the training and test datasets.

The train_test_split function helps with this.

```
from sklearn.model_selection import
  train_test_split
```

Train_test_split returns a tuple with 4 elements: the training features, the testing features, the training targets and the testing targets.

scikit-learn

Popular datasets (like iris) are included with scikit_learn

```
from sklearn.datasets import load_iris
iris = load_iris()
```

The features are in the data field

```
features = iris.data
```

The targets are in the target field

```
targets = iris.target
```

A selection of the data will be reserved for testing the model after it is trained. It is important to ensure that the targets are equally represented in both the training and test datasets.

The train_test_split function helps with this.

```
from sklearn.model_selection import
  train_test_split
```

Train_test_split returns a tuple with 4 elements: the training features, the testing features, the training targets and the testing targets.

```
X_train, X_test, y_train, y_test =
  train_test_split(features, targets)
```

scikit-learn

And specify the size of each.

scikit-learn

And specify the size of each.

```
X_train, X_test, y_train, y_test =
  train_test_split(features, targets,
  train_size=0.6, test_size=0.4, random_state=3824)
```

scikit-learn

And specify the size of each.

To train the model, import the KNeighborsClassifier

```
X_train, X_test, y_train, y_test =
  train_test_split(features, targets,
  train_size=0.6, test_size=0.4, random_state=3824)
```

scikit-learn

And specify the size of each.

```
X_train, X_test, y_train, y_test =
  train_test_split(features, targets,
  train_size=0.6, test_size=0.4, random_state=3824)
```

To train the model, import the KNeighborsClassifier

```
from sklearn.neighbors import KNeighborsClassifier
```

scikit-learn

And specify the size of each.

```
X_train, X_test, y_train, y_test =
  train_test_split(features, targets,
  train_size=0.6, test_size=0.4, random_state=3824)
```

To train the model, import the KNeighborsClassifier

```
from sklearn.neighbors import KNeighborsClassifier
```

Create a new classifier to train the model. As a hyperparameter, set the number of neighbors to 3.

scikit-learn

And specify the size of each.

```
X_train, X_test, y_train, y_test =
  train_test_split(features, targets,
  train_size=0.6, test_size=0.4, random_state=3824)
```

To train the model, import the KNeighborsClassifier

```
from sklearn.neighbors import KNeighborsClassifier
```

Create a new classifier to train the model. As a hyperparameter, set the number of neighbors to 3.

```
model = KNeighborsClassifier(n_neighbors=3)
```

scikit-learn

And specify the size of each.

```
X_train, X_test, y_train, y_test =
  train_test_split(features, targets,
  train_size=0.6, test_size=0.4, random_state=3824)
```

To train the model, import the KNeighborsClassifier

```
from sklearn.neighbors import KNeighborsClassifier
```

Create a new classifier to train the model. As a hyperparameter, set the number of neighbors to 3.

```
model = KNeighborsClassifier(n_neighbors=3)
```

The fit method will train the model. It accepts the training features and training targets.

scikit-learn

And specify the size of each.

```
X_train, X_test, y_train, y_test =
  train_test_split(features, targets,
  train_size=0.6, test_size=0.4, random_state=3824)
```

To train the model, import the KNeighborsClassifier

```
from sklearn.neighbors import KNeighborsClassifier
```

Create a new classifier to train the model. As a hyperparameter, set the number of neighbors to 3.

```
model = KNeighborsClassifier(n_neighbors=3)
```

The fit method will train the model. It accepts the training features and training targets.

```
model.fit(X_train, y_train)
```

scikit-learn

And specify the size of each.

```
X_train, X_test, y_train, y_test =
  train_test_split(features, targets,
  train_size=0.6, test_size=0.4, random_state=3824)
```

To train the model, import the KNeighborsClassifier

```
from sklearn.neighbors import KNeighborsClassifier
```

Create a new classifier to train the model. As a hyperparameter, set the number of neighbors to 3.

```
model = KNeighborsClassifier(n_neighbors=3)
```

The fit method will train the model. It accepts the training features and training targets.

```
model.fit(X_train, y_train)
```

Make predictions using the testing features

scikit-learn

And specify the size of each.

```
X_train, X_test, y_train, y_test =
  train_test_split(features, targets,
  train_size=0.6, test_size=0.4, random_state=3824)
```

To train the model, import the KNeighborsClassifier

```
from sklearn.neighbors import KNeighborsClassifier
```

Create a new classifier to train the model. As a hyperparameter, set the number of neighbors to 3.

```
model = KNeighborsClassifier(n_neighbors=3)
```

The fit method will train the model. It accepts the training features and training targets.

```
model.fit(X_train, y_train)
```

Make predictions using the testing features

```
predictions = model.predict(X_test)
```

scikit-learn

And specify the size of each.

```
X_train, X_test, y_train, y_test =
  train_test_split(features, targets,
  train_size=0.6, test_size=0.4, random_state=3824)
```

To train the model, import the KNeighborsClassifier

```
from sklearn.neighbors import KNeighborsClassifier
```

Create a new classifier to train the model. As a hyperparameter, set the number of neighbors to 3.

```
model = KNeighborsClassifier(n_neighbors=3)
```

The fit method will train the model. It accepts the training features and training targets.

```
model.fit(X_train, y_train)
```

Make predictions using the testing features

```
predictions = model.predict(X_test)
```

The score method will compute the overall performance of the predictions

scikit-learn

And specify the size of each.

```
X_train, X_test, y_train, y_test =
  train_test_split(features, targets,
  train_size=0.6, test_size=0.4, random_state=3824)
```

To train the model, import the KNeighborsClassifier

```
from sklearn.neighbors import KNeighborsClassifier
```

Create a new classifier to train the model. As a hyperparameter, set the number of neighbors to 3.

```
model = KNeighborsClassifier(n_neighbors=3)
```

The fit method will train the model. It accepts the training features and training targets.

```
model.fit(X_train, y_train)
```

Make predictions using the testing features

```
predictions = model.predict(X_test)
```

The score method will compute the overall performance of the predictions

```
model.score(X_test, y_test)
```

scikit-learn

Not bad, but the model had some errors.

scikit-learn

Not bad, but the model had some errors.

A “confusion matrix” will show what errors were made and where.

scikit-learn

Not bad, but the model had some errors.

A “confusion matrix” will show what errors were made and where.

Scikit-learn has a function, confusion_matrix, for this purpose

scikit-learn

Not bad, but the model had some errors.

A “confusion matrix” will show what errors were made and where.

Scikit-learn has a function, confusion_matrix, for this purpose

```
from sklearn.metrics import confusion_matrix
```

scikit-learn

Not bad, but the model had some errors.

A “confusion matrix” will show what errors were made and where.

Scikit-learn has a function, confusion_matrix, for this purpose

```
from sklearn.metrics import confusion_matrix
```

The function accepts the “true” targets (from the testing data set, and the predicted targets.

scikit-learn

Not bad, but the model had some errors.

A “confusion matrix” will show what errors were made and where.

Scikit-learn has a function, confusion_matrix, for this purpose

```
from sklearn.metrics import confusion_matrix
```

The function accepts the “true” targets (from the testing data set, and the predicted targets.

```
cm = confusion_matrix(y_test, predictions)
```

scikit-learn

Not bad, but the model had some errors.

A “confusion matrix” will show what errors were made and where.

Scikit-learn has a function, confusion_matrix, for this purpose

```
from sklearn.metrics import confusion_matrix
```

The function accepts the “true” targets (from the testing data set, and the predicted targets.

```
cm = confusion_matrix(y_test, predictions)
```

The ConfusionMatrixDisplay uses matplotlib to graphically illustrate the errors. (The “gray” cmap, or color map, overrides the default yellow/green/blue color scheme)

scikit-learn

Not bad, but the model had some errors.

A “confusion matrix” will show what errors were made and where.

Scikit-learn has a function, confusion_matrix, for this purpose

```
from sklearn.metrics import confusion_matrix
```

The function accepts the “true” targets (from the testing data set, and the predicted targets.

```
cm = confusion_matrix(y_test, predictions)
```

The ConfusionMatrixDisplay uses matplotlib to graphically illustrate the errors. (The “gray” cmap, or color map, overrides the default yellow/green/blue color scheme)

```
from sklearn.metrics import ConfusionMatrixDisplay
cmd = ConfusionMatrixDisplay(
  confusion_matrix=cm,
  display_labels=iris.target_names)
cmd.plot(cmap=“gray”)
```

scikit-learn

The model did great with setosa, but not as much with the other classes.

scikit-learn

The model did great with setosa, but not as much with the other classes.

Create a scatter plot, with the first two features, using the class to color them. To get the legend, each class must be plotted separately and labeled with the target name.

scikit-learn

The model did great with setosa, but not as much with the other classes.

Create a scatter plot, with the first two features, using the class to color them. To get the legend, each class must be plotted separately and labeled with the target name.

```
import matplotlib.pyplot as plt
colors = list("rgb")
for idx, target in enumerate(iris.target_names):
  subset = features[targets == idx]
  plt.scatter(subset[:, 0], subset[:, 1], label=target, c=colors[idx])
plt.legend()
```

scikit-learn

The model did great with setosa, but not as much with the other classes.

Create a scatter plot, with the first two features, using the class to color them. To get the legend, each class must be plotted separately and labeled with the target name.

```
import matplotlib.pyplot as plt
colors = list(“rgb”)
for idx, target in enumerate(iris.target_names):
  subset = features[targets == idx]
  plt.scatter(subset[:, 0], subset[:, 1], label=target, c=colors[idx])
plt.legend()
```

It looks like setosa, is clearly separated from the other two classes. Also, versicolor and virginica are overlapping. This could mean its easier for the model to define the space for setosa.

scikit-learn

The model did great with setosa, but not as much with the other classes.

Create a scatter plot, with the first two features, using the class to color them. To get the legend, each class must be plotted separately and labeled with the target name.

```
import matplotlib.pyplot as plt
colors = list(“rgb”)
for idx, target in enumerate(iris.target_names):
  subset = features[targets == idx]
  plt.scatter(subset[:, 0], subset[:, 1], label=target, c=colors[idx])
plt.legend()
```

It looks like setosa, is clearly separated from the other two classes. Also, versicolor and virginica are overlapping. This could mean its easier for the model to define the space for setosa.

But this is just two of the four features. We need to consider all of the combinations.

scikit-learn

A scatter matrix will create a grid, with the features on the rows and columns, and create a scatter plot for each combination

scikit-learn

A scatter matrix will create a grid, with the features on the rows and columns, and create a scatter plot for each combination

The seaborn package include a function to do this for us. (but it's named pairplot)

scikit-learn

A scatter matrix will create a grid, with the features on the rows and columns, and create a scatter plot for each combination

The seaborn package include a function to do this for us. (but it’s named pairplot)

Before using the pairplot, we need to put the data in a DataFrame

scikit-learn

A scatter matrix will create a grid, with the features on the rows and columns, and create a scatter plot for each combination

The seaborn package include a function to do this for us. (but it's named pairplot)

Before using the pairplot, we need to put the data in a DataFrame

```
import pandas as pd
iris_df = pd.DataFrame(data=features, columns=iris.feature_names)
iris_df["species"] = targets
iris_class_dict = dict([(idx, class_) for idx, class_ in enumerate(iris.target_names)])
iris_df["species"] = iris_df["species"].apply(lambda x: iris_classes[x])
```

scikit-learn

Install the seaborn package

scikit-learn

Install the seaborn package

```
%pip install seaborn
```

scikit-learn

Install the seaborn package

```
%pip install seaborn
```

Import the seaborn module and use sns as the alias

scikit-learn

Install the seaborn package

```
%pip install seaborn
```

Import the seaborn module and use sns as the alias

```
import seaborn as sns
```

scikit-learn

Install the seaborn package

```
%pip install seaborn
```

Import the seaborn module and use sns as the alias

```
import seaborn as sns
```

Call the pairplot function and pass it the DataFrame.

scikit-learn

Install the seaborn package	`%pip install seaborn`
Import the seaborn module and use sns as the alias	`import seaborn as sns`
Call the pairplot function and pass it the DataFrame.	`sns.pairplot(iris_df, diag=None)`

scikit-learn

Install the seaborn package

```
%pip install seaborn
```

Import the seaborn module and use sns as the alias

```
import seaborn as sns
```

Call the pairplot function and pass it the DataFrame.

```
sns.pairplot(iris_df, diag=None)
```

The hue keyword will use a column from the DataFrame to as the colors.

scikit-learn

Install the seaborn package	`%pip install seaborn`
Import the seaborn module and use sns as the alias	`import seaborn as sns`
Call the pairplot function and pass it the DataFrame.	`sns.pairplot(iris_df, diag=None)`
The hue keyword will use a column from the DataFrame to as the colors.	`sns.pairplot(iris_df, diag=None, hue=“species”)`

Machine Learning

What is the goal of deep learning?

Machine Learning

What is the goal of deep learning?

The goal of machine deep is to
mimic human response ...

Machine Learning

What is the goal of deep learning?

The goal of machine deep is to
mimic human response ...

without being explicitly programmed ...

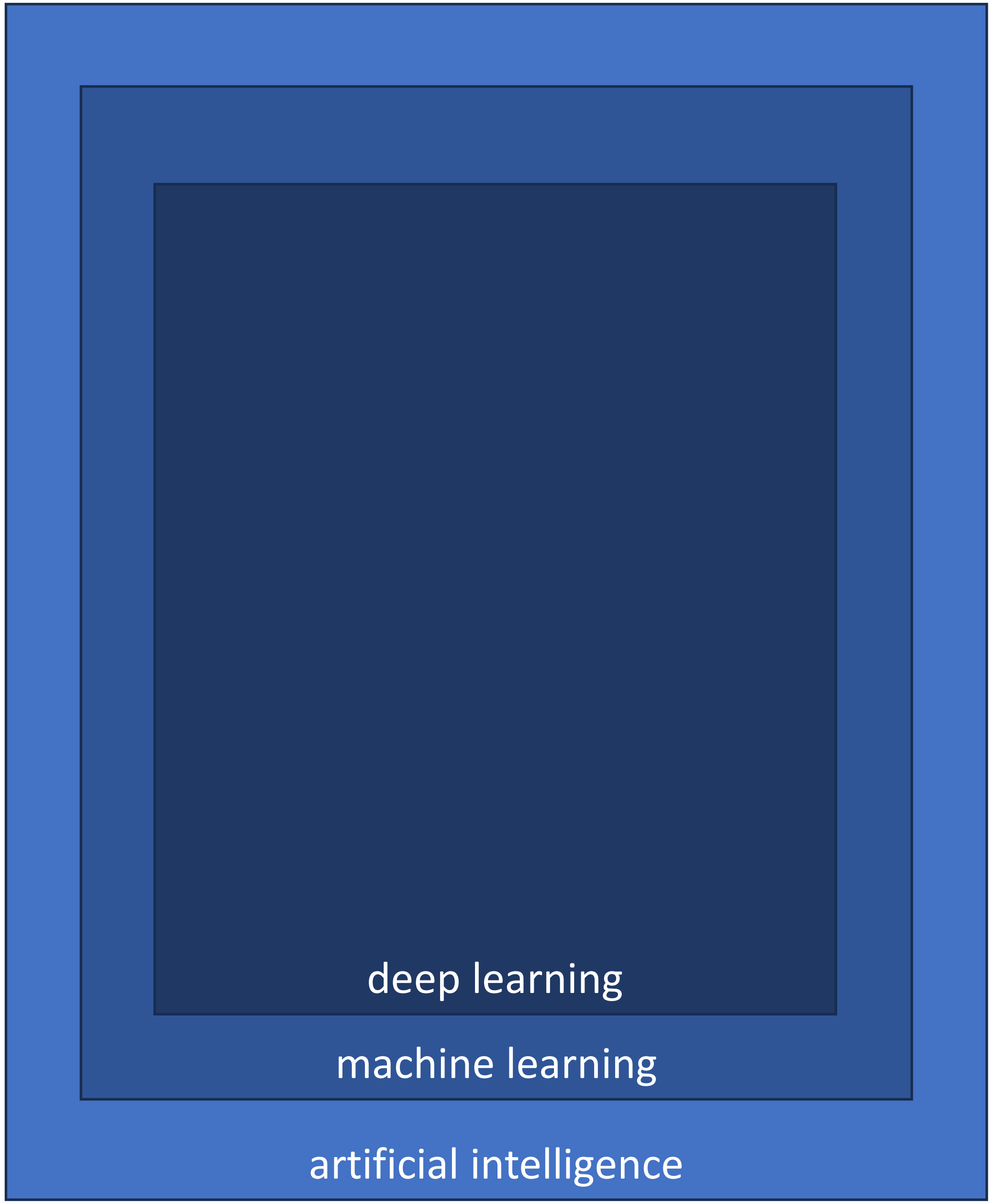
Machine Learning

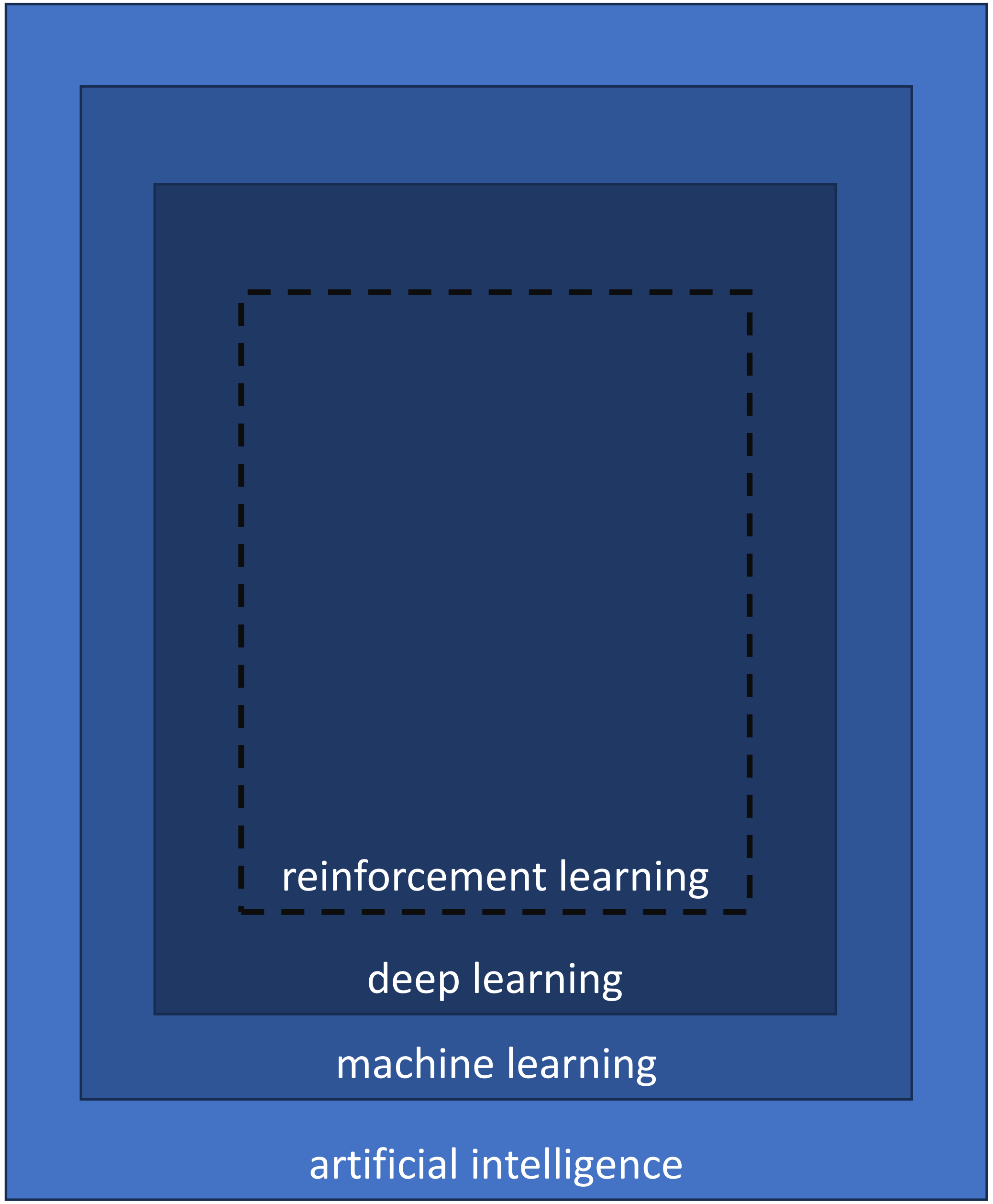
What is the goal of deep learning?

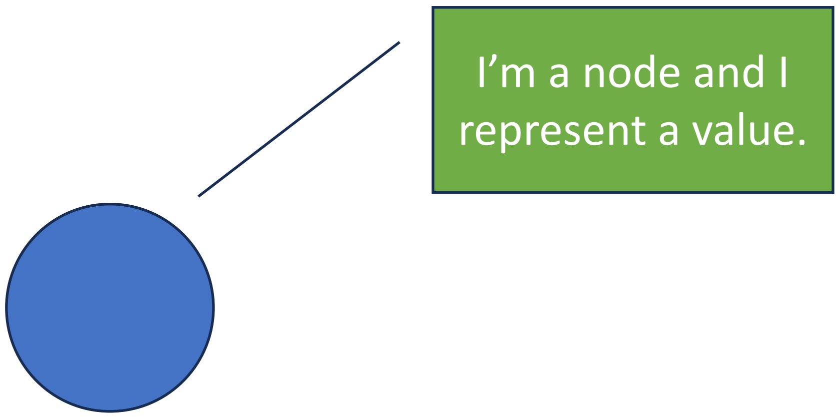
The goal of machine deep is to
mimic human response ...

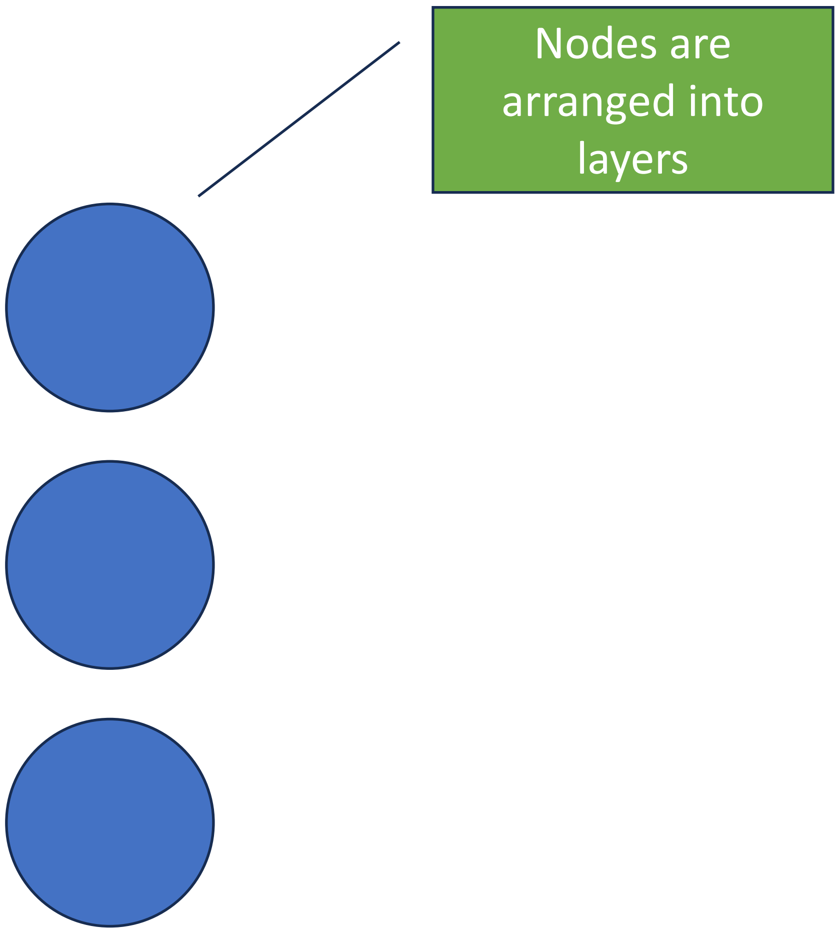
without being explicitly programmed ...

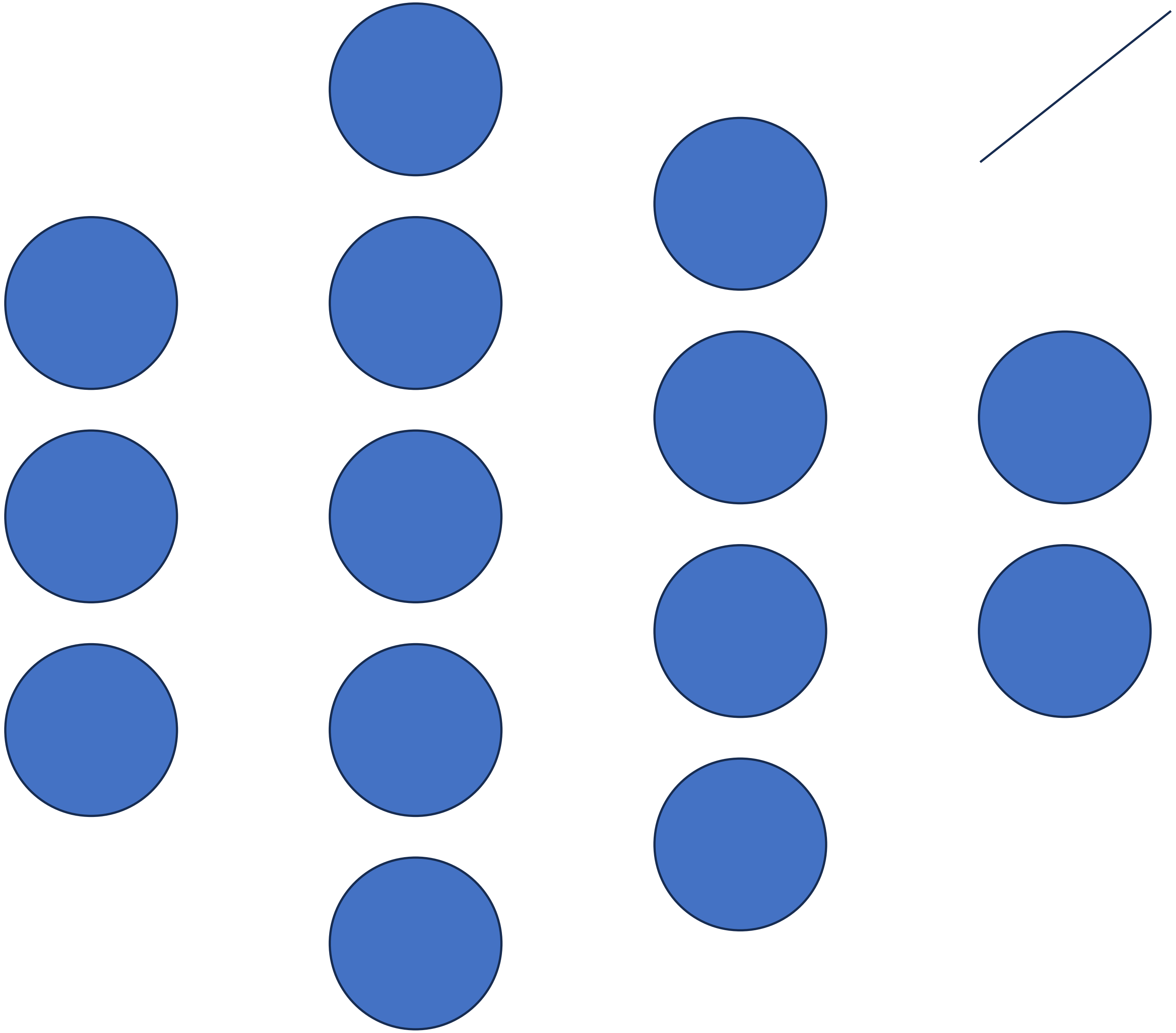
using neural networks.


deep learning
machine learning
artificial intelligence

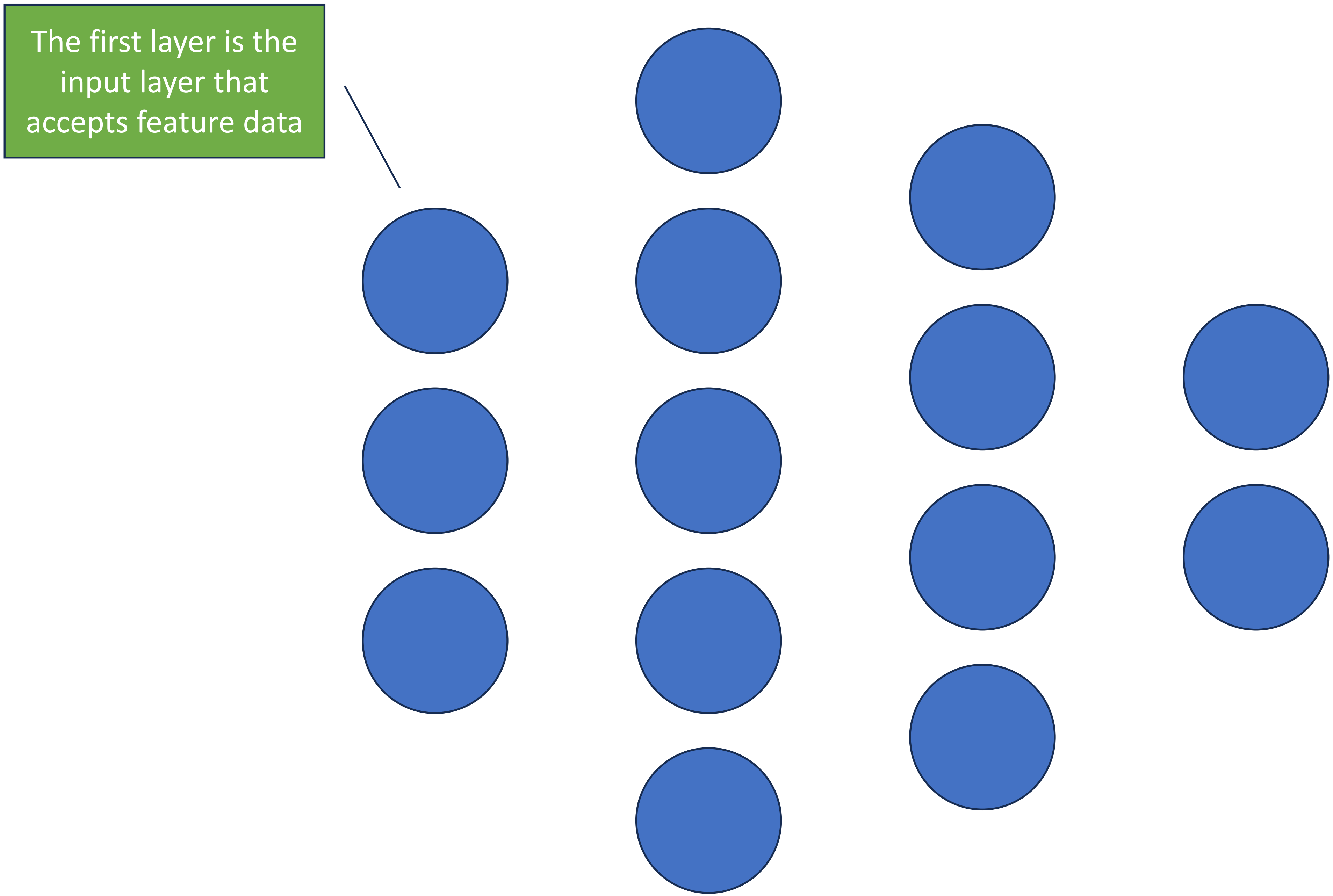

reinforcement learning
deep learning
machine learning
artificial intelligence

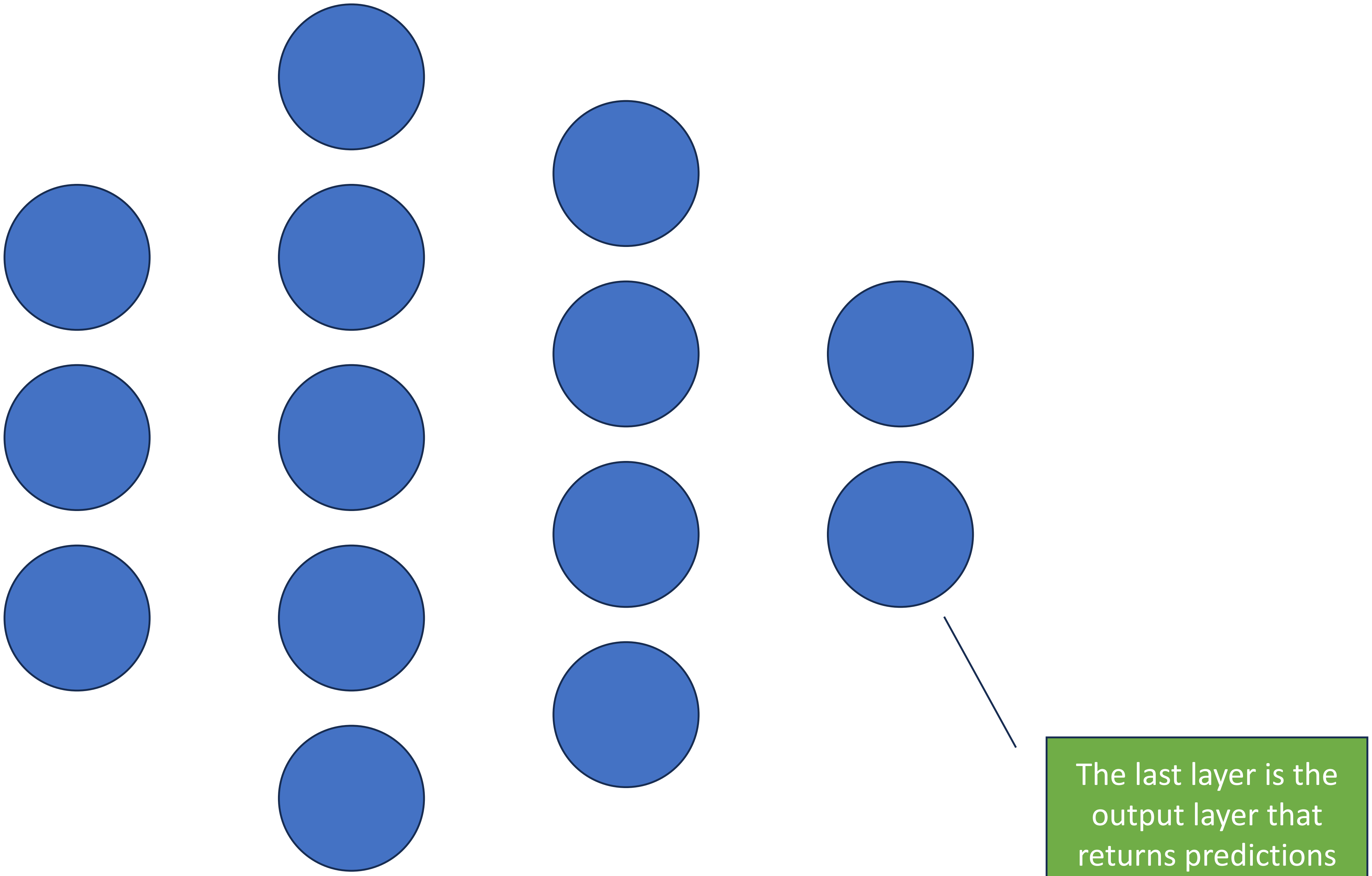

I’m a node and I represent a value.

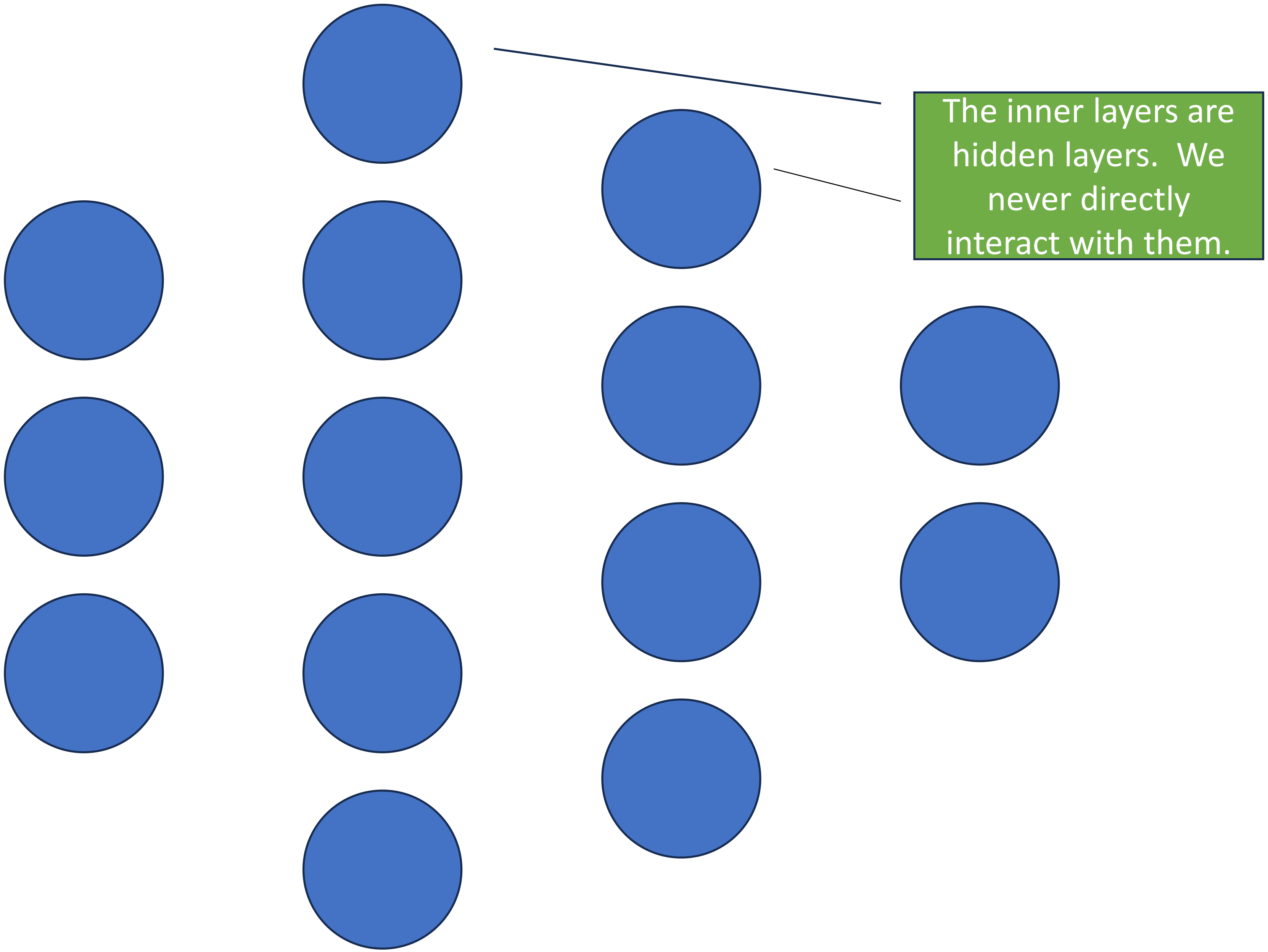

Nodes are arranged into layers

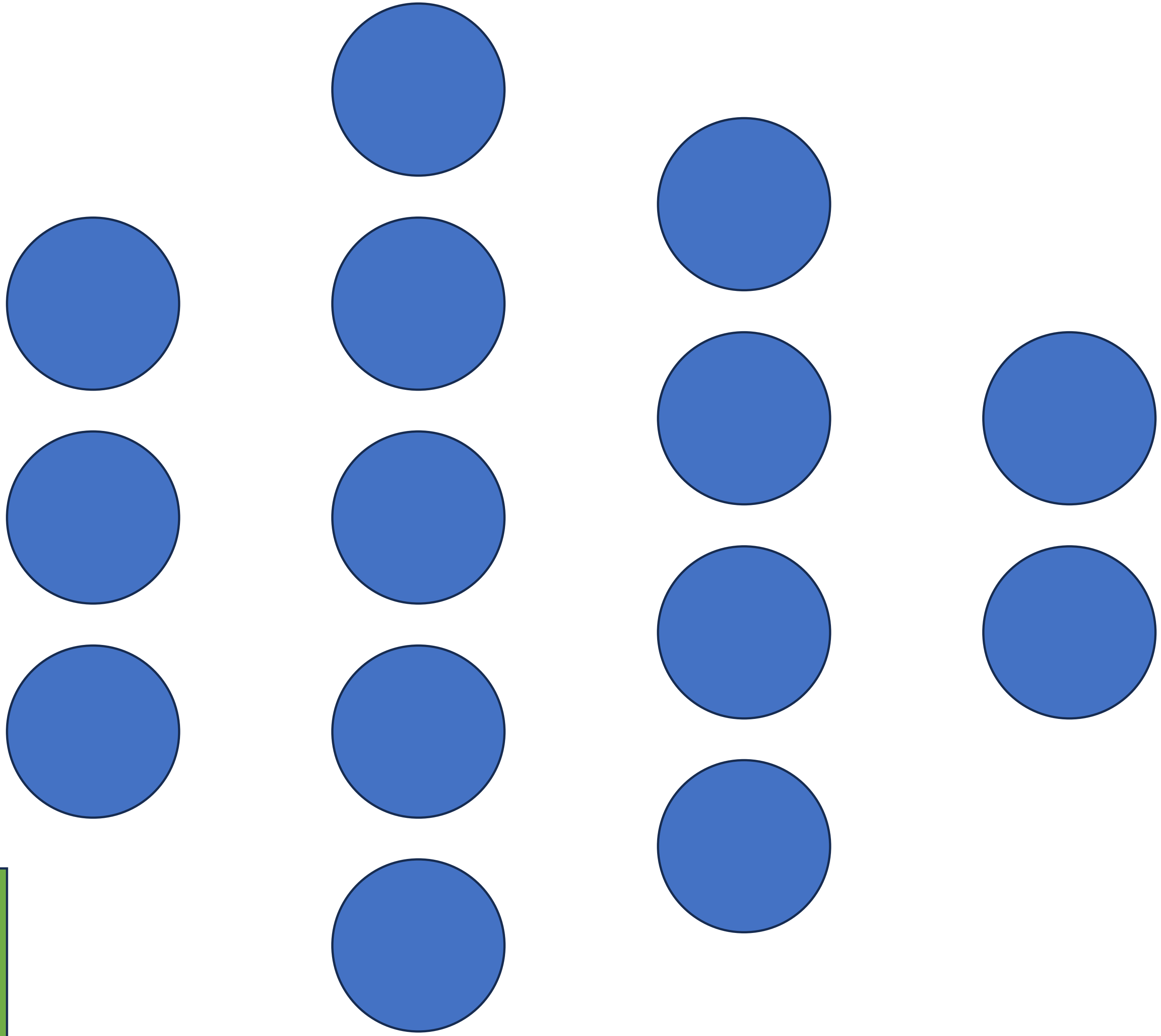


A neural network is made up of a sequence of layers.

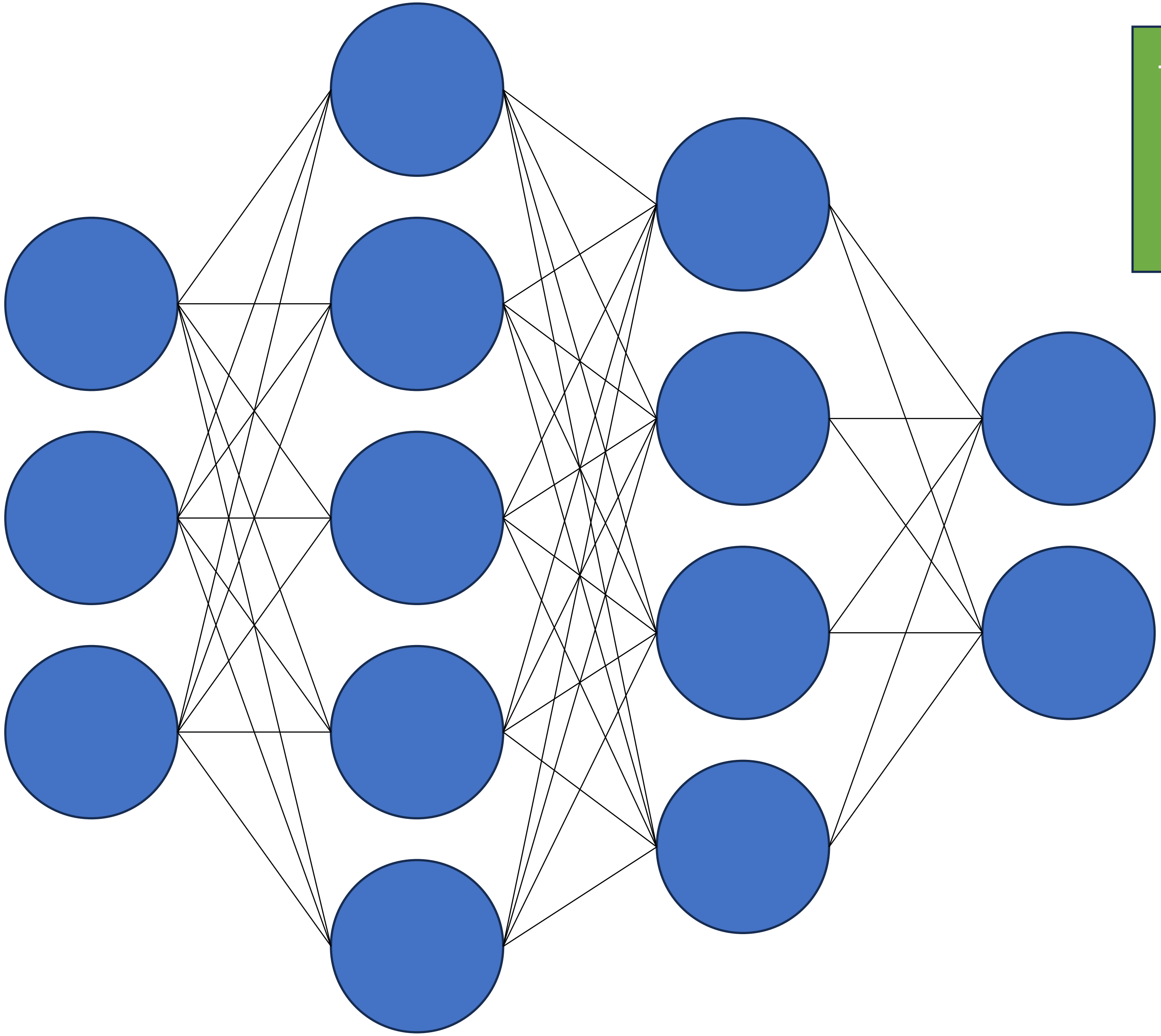

The first layer is the input layer that accepts feature data


The last layer is the output layer that returns predictions

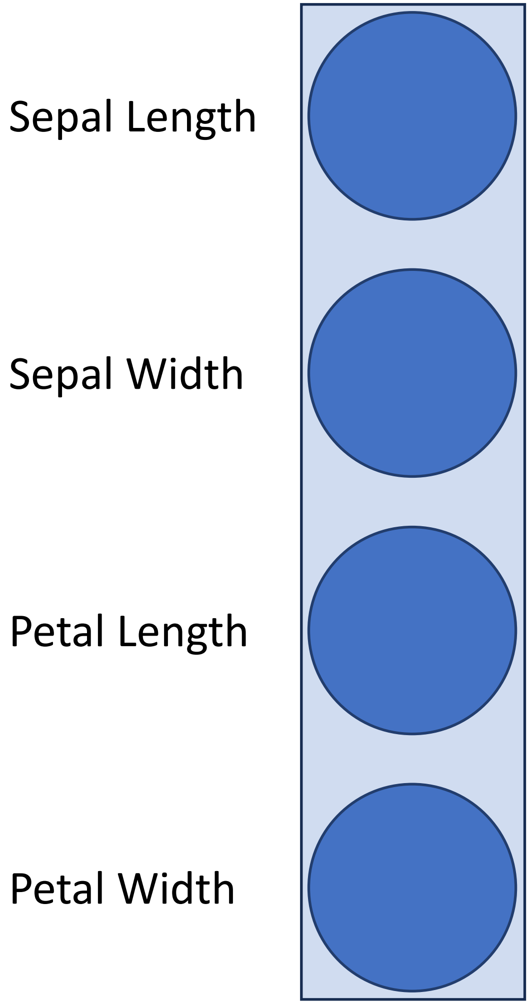

The inner layers are hidden layers. We never directly interact with them.

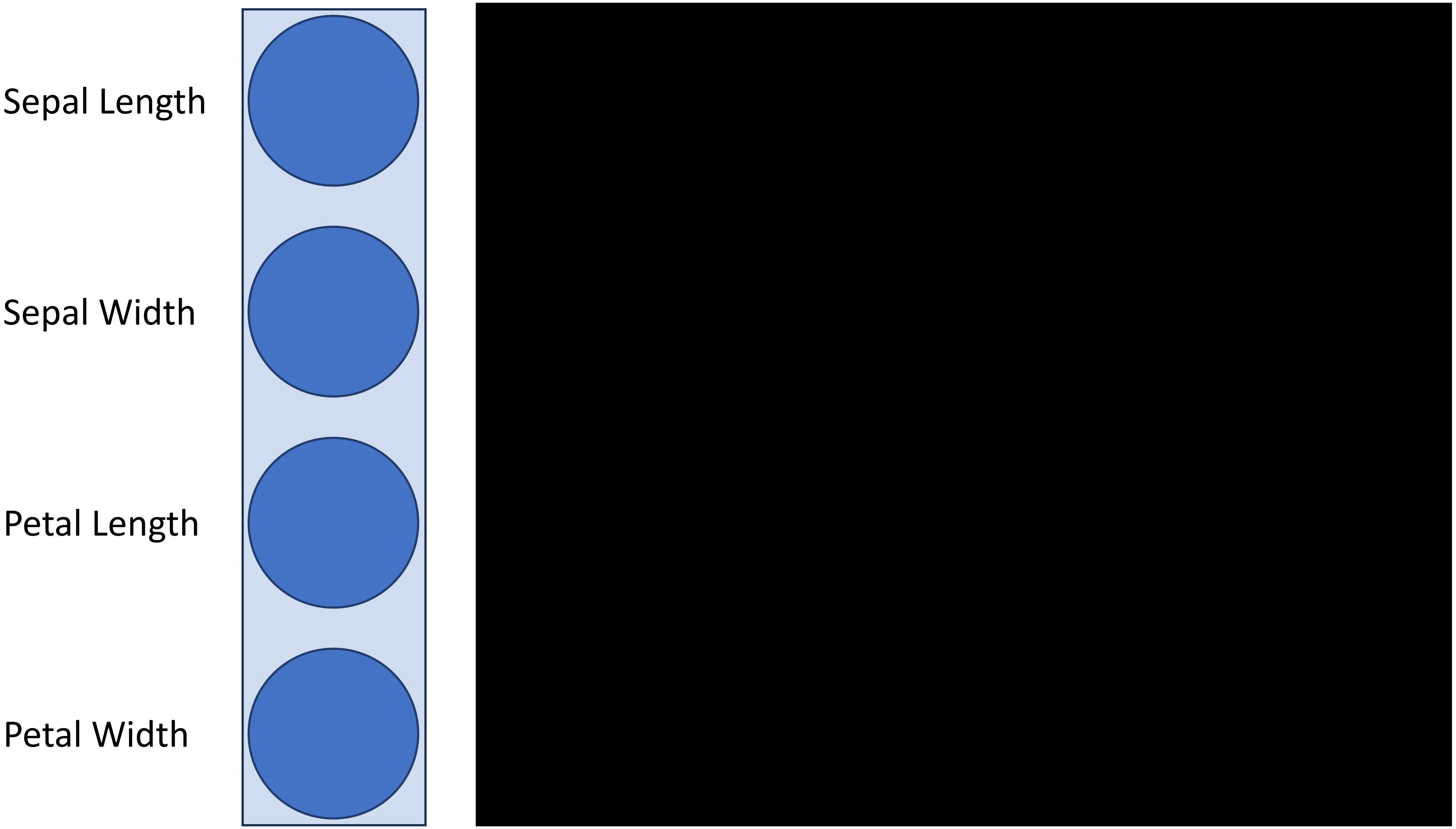


A neural network with more than 2 hidden layers is a deep neural network

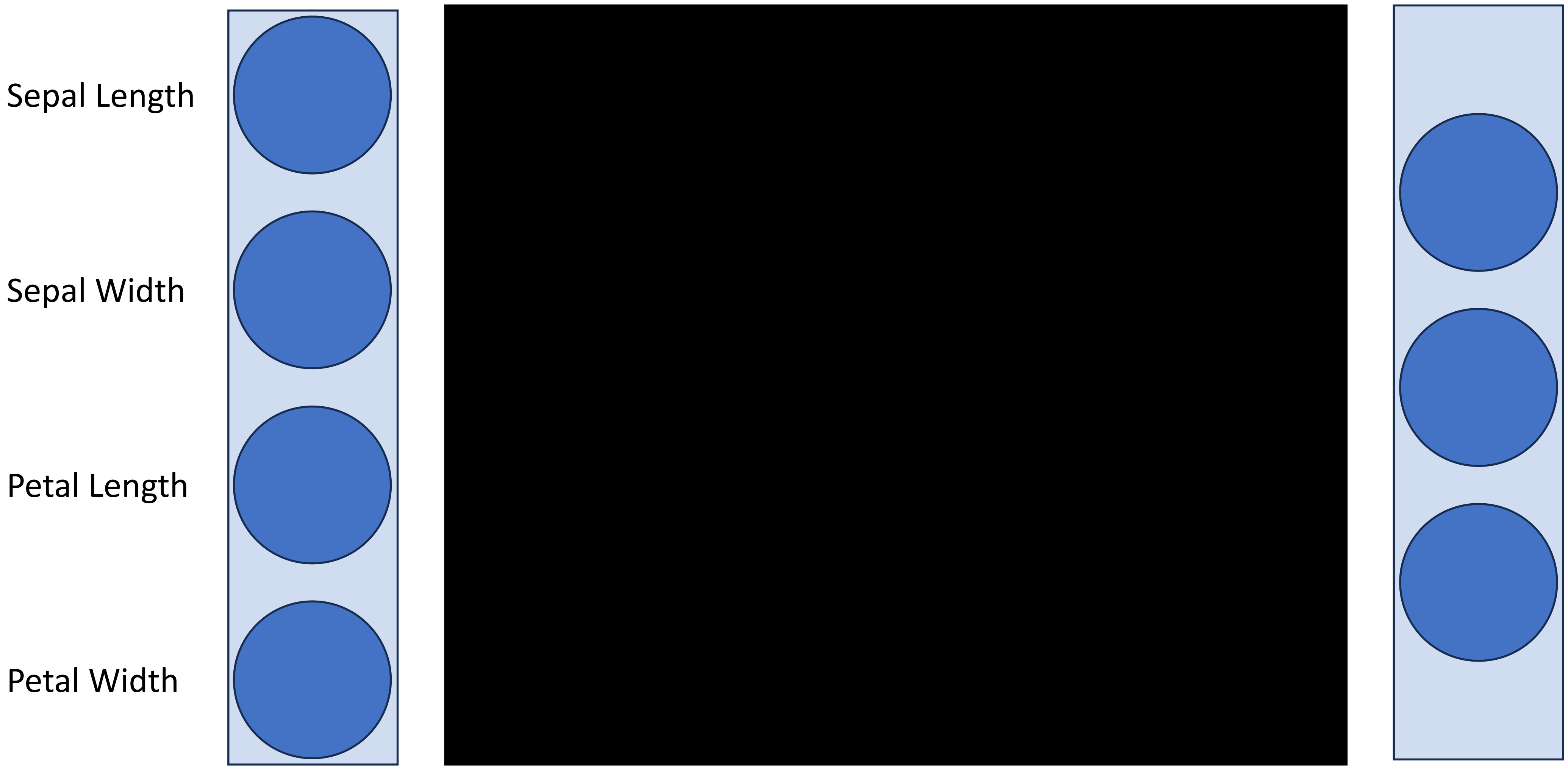


This is a dense network. All nodes in one layer are connected to all nodes in the next.


Sepal Length
Sepal Width
Petal Length
Petal Width


Sepal Length
Sepal Width
Petal Length
Petal Width


Sepal Length
Sepal Width
Petal Length
Petal Width

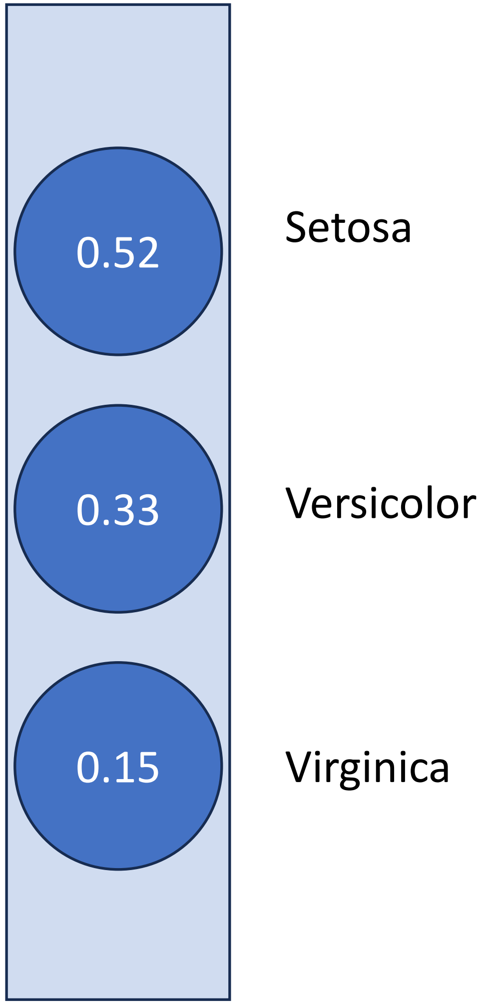

Sepal Length
Sepal Width
Petal Length
Petal Width


Sepal Length
Sepal Width
Petal Length
Petal Width
Setosa
Versicolor
Virginica

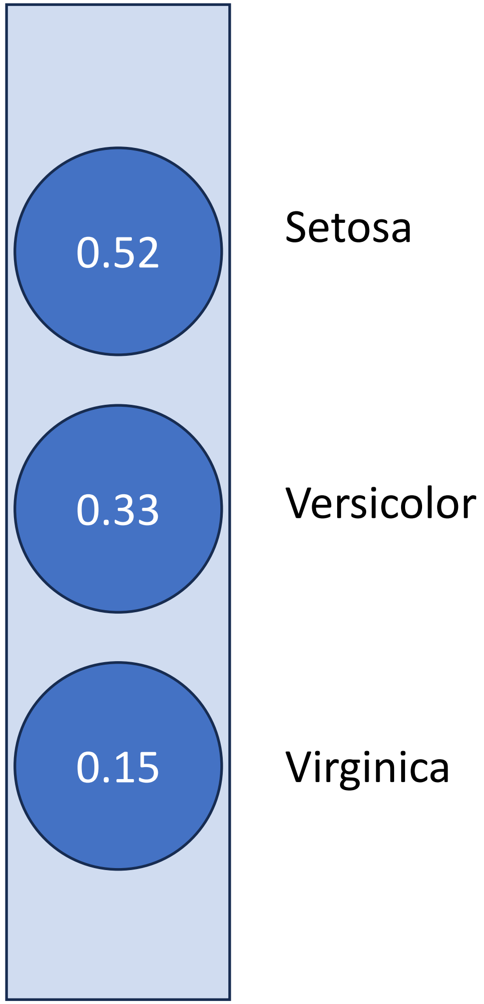

Sepal Length
Sepal Width
Petal Length
Petal Width
0.52
0.33
0.15
Setosa
Versicolor
Virginica

Sepal Length	Sepal Width	Petal Length	Petal Width	Species
5.1	3.5	1.4	0.2	Setosa

Sepal Length	Sepal Width	Petal Length	Petal Width	Species
5.1	3.5	1.4	0.2	Setosa

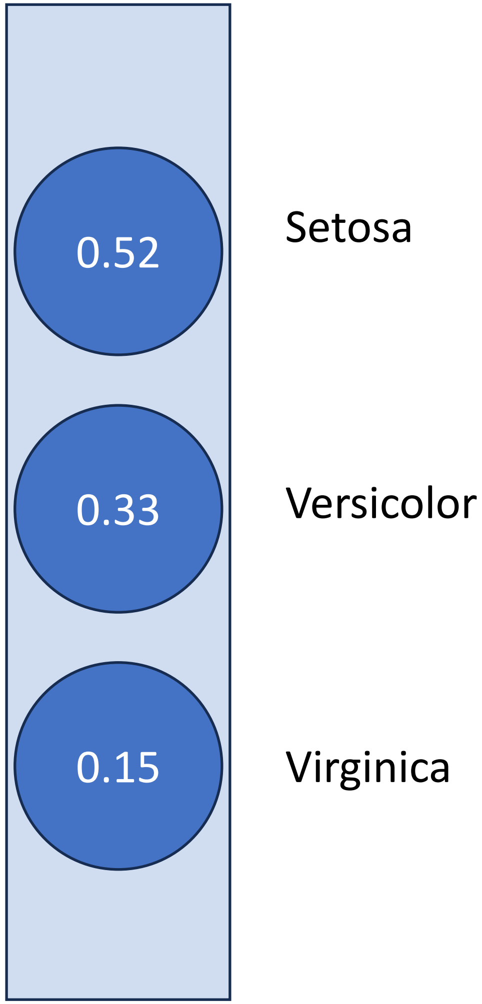


Sepal Length	Sepal Width	Petal Length	Petal Width	Species
5.1	3.5	1.4	0.2	Setosa



Who thinks this prediction is correct?

Sepal Length	Sepal Width	Petal Length	Petal Width	Species
5.1	3.5	1.4	0.2	Setosa



Who thinks this prediction is correct?

Who thinks this prediction is incorrect?

Sepal Length	Sepal Width	Petal Length	Petal Width	Species
5.1	3.5	1.4	0.2	Setosa

0.52 Setosa

0.33 Versicolor

0.15 Virginica

Who thinks this prediction is correct?

Who thinks this prediction is incorrect?

You’re all wrong!

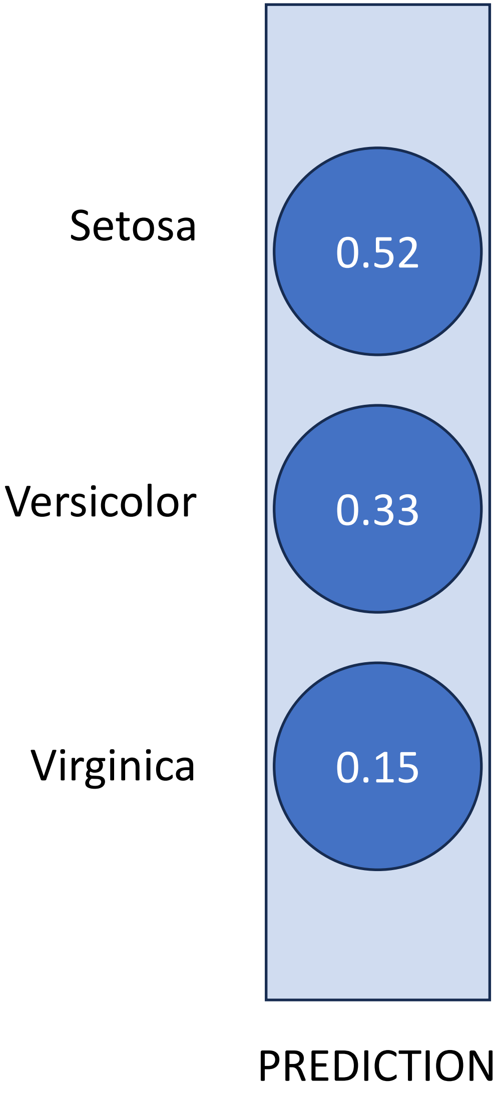

who
cares?
11 12 1
10 2 3 4
9 8 7 6 5

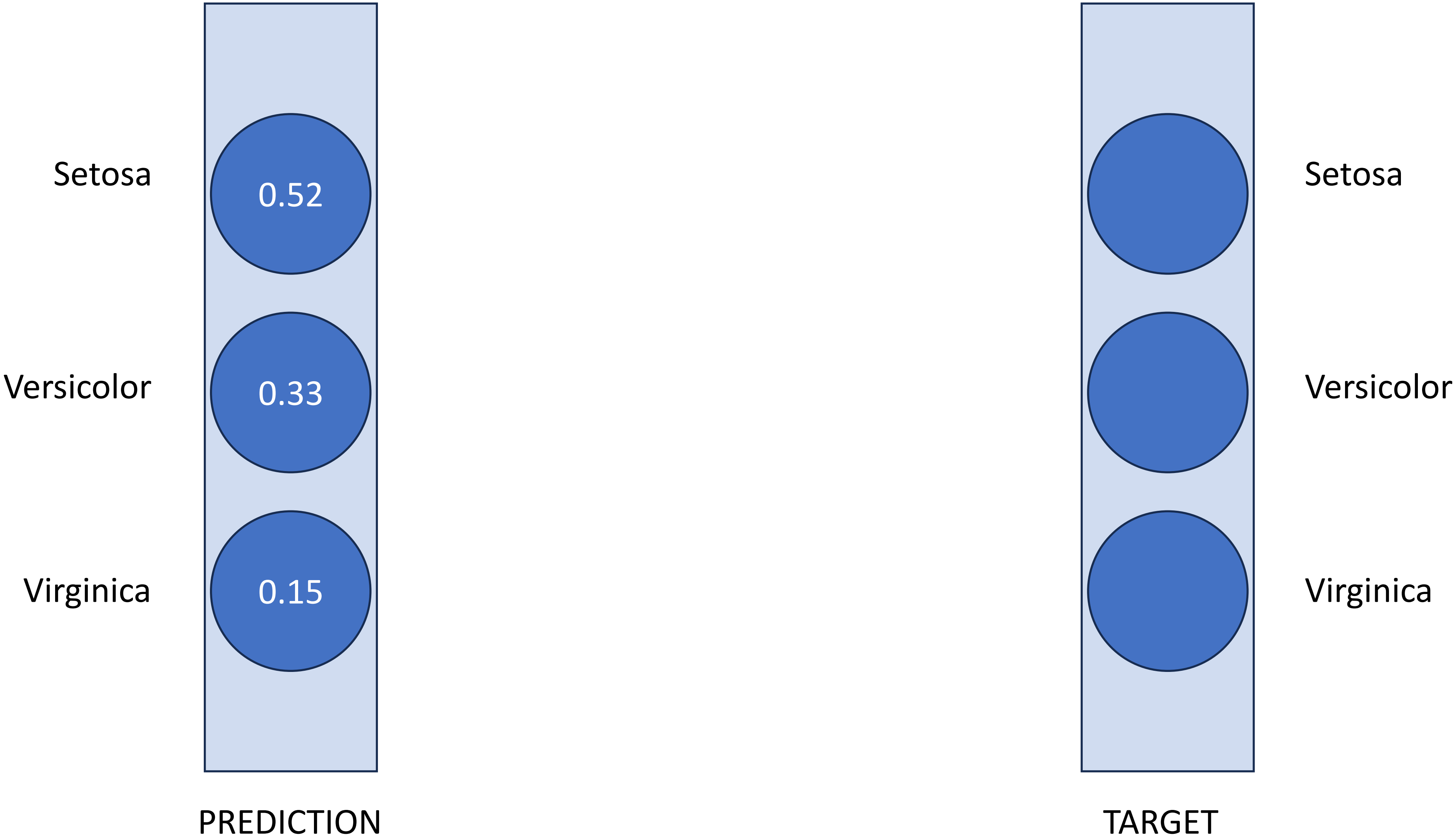

who
cares?

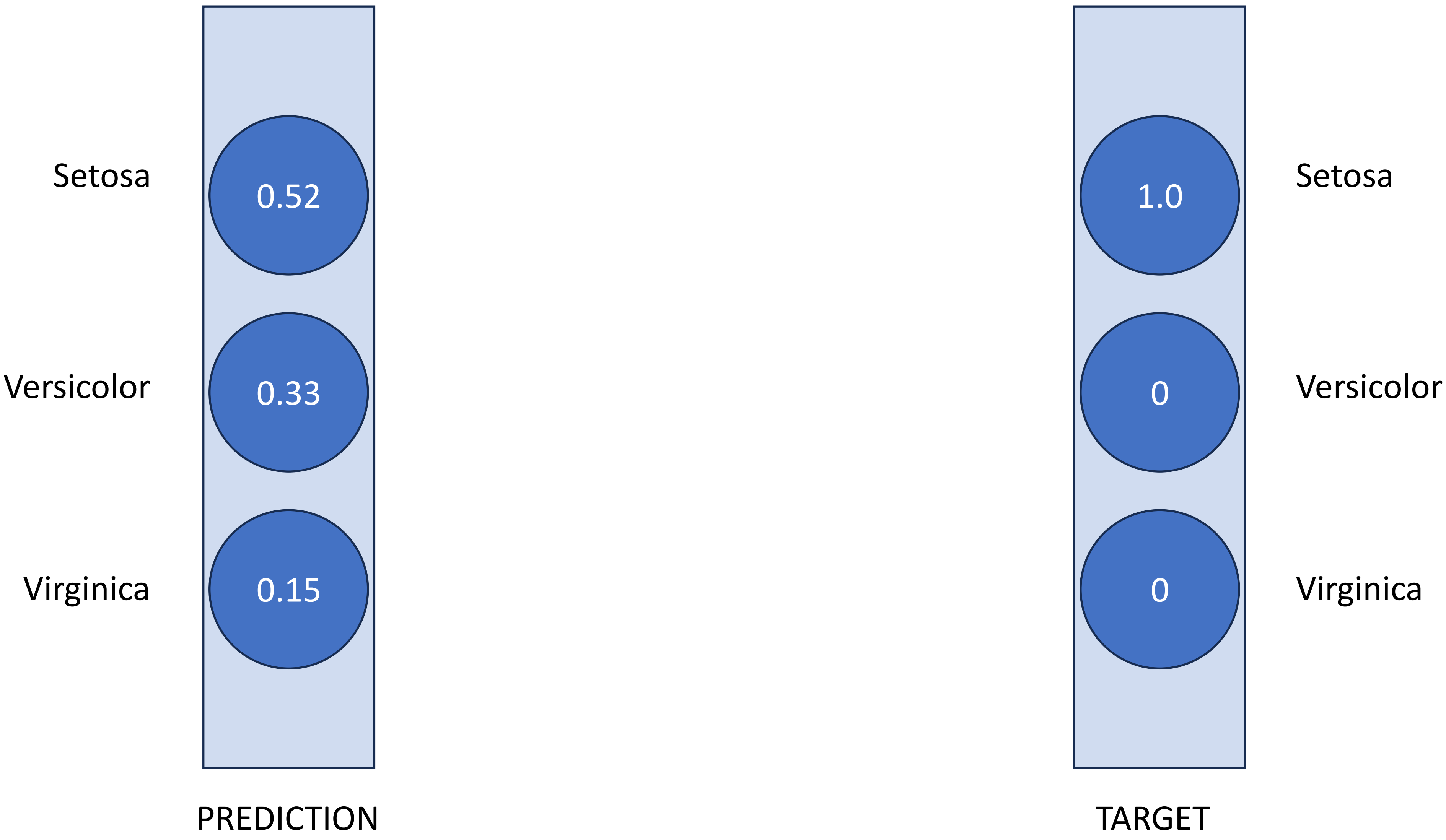

Don’t worry, no more trick questions in the workshop!

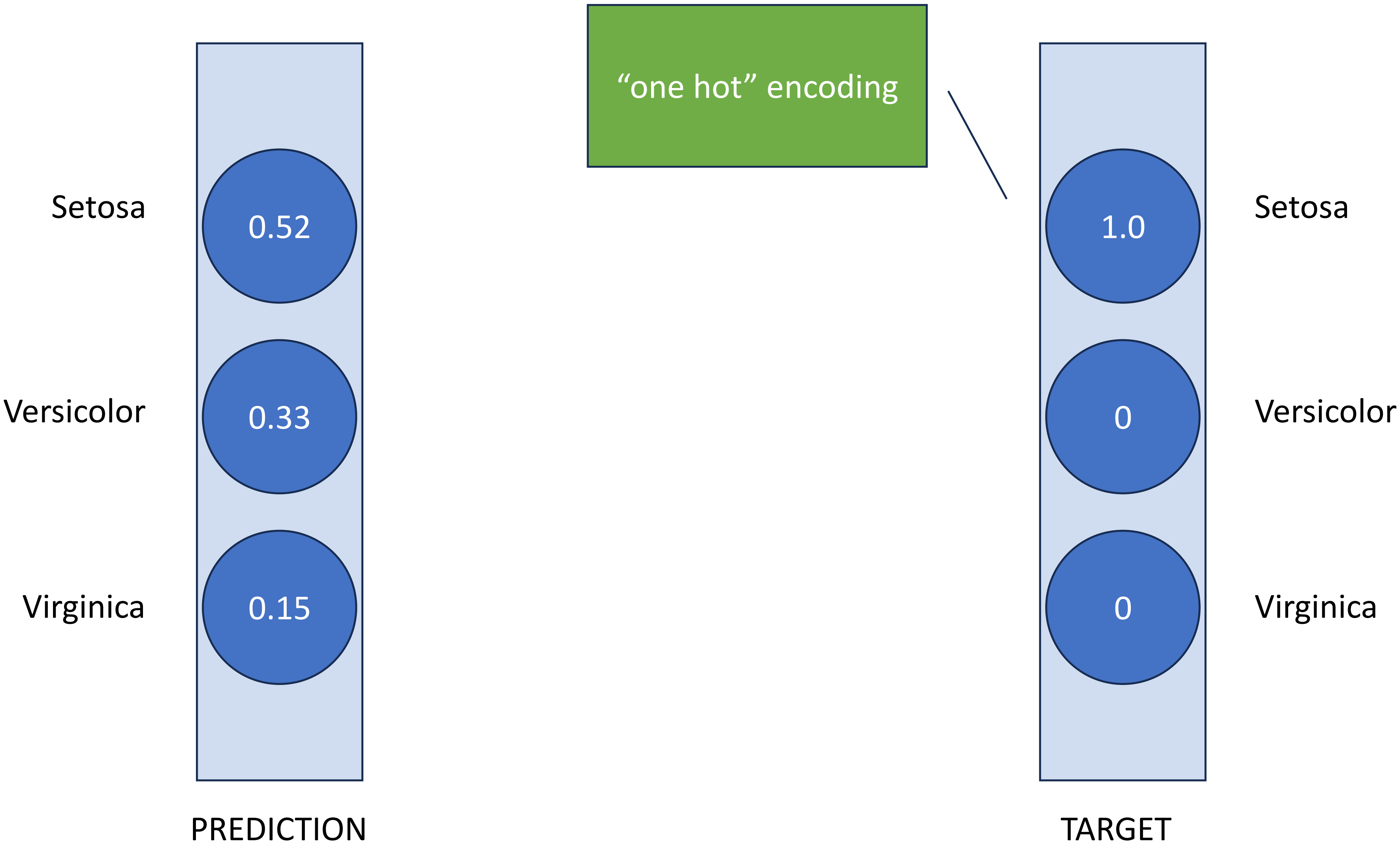
During training, we are not concerned with making correct predictions but predictions that are as close to correct as possible.

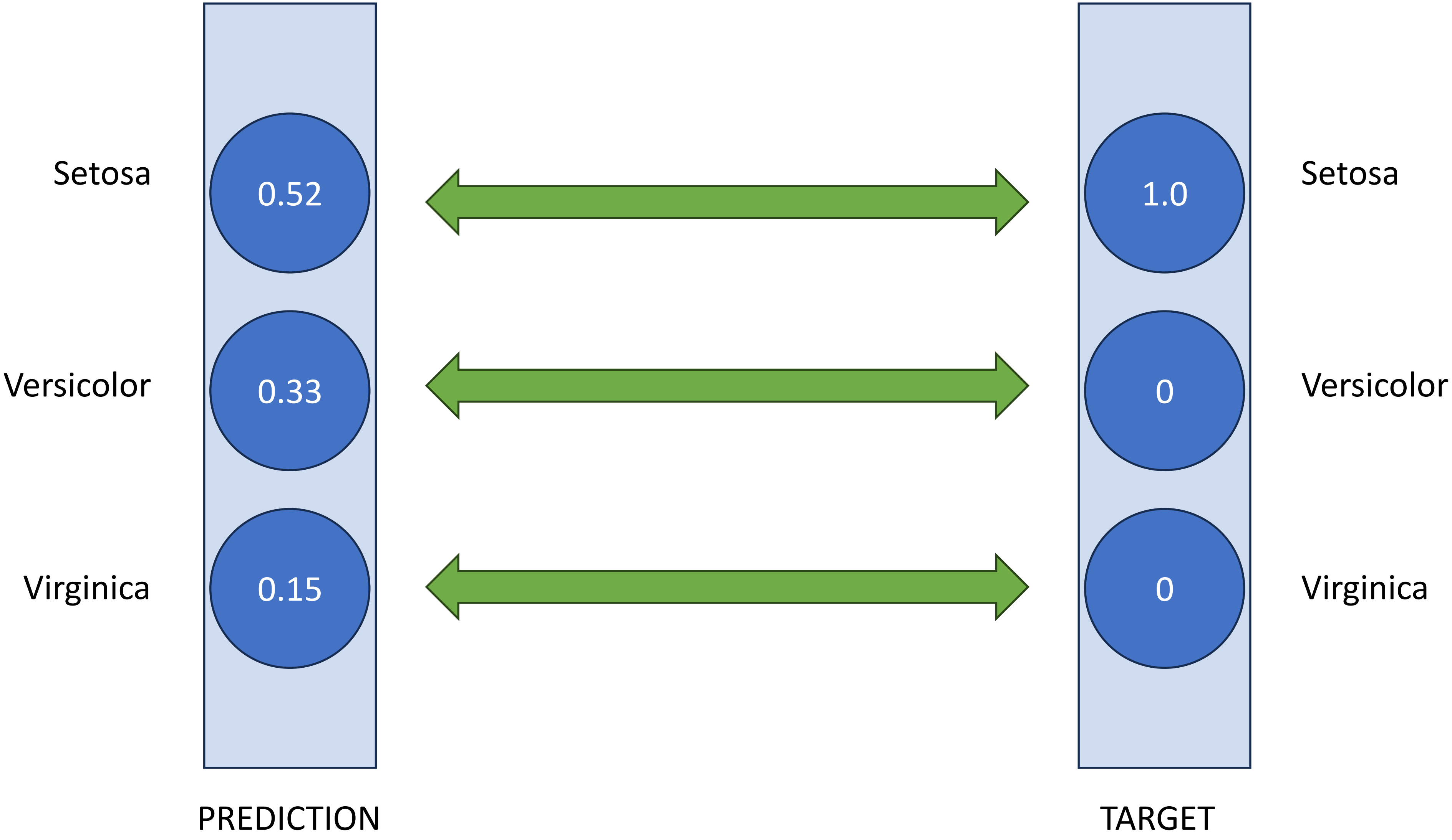

WTF?

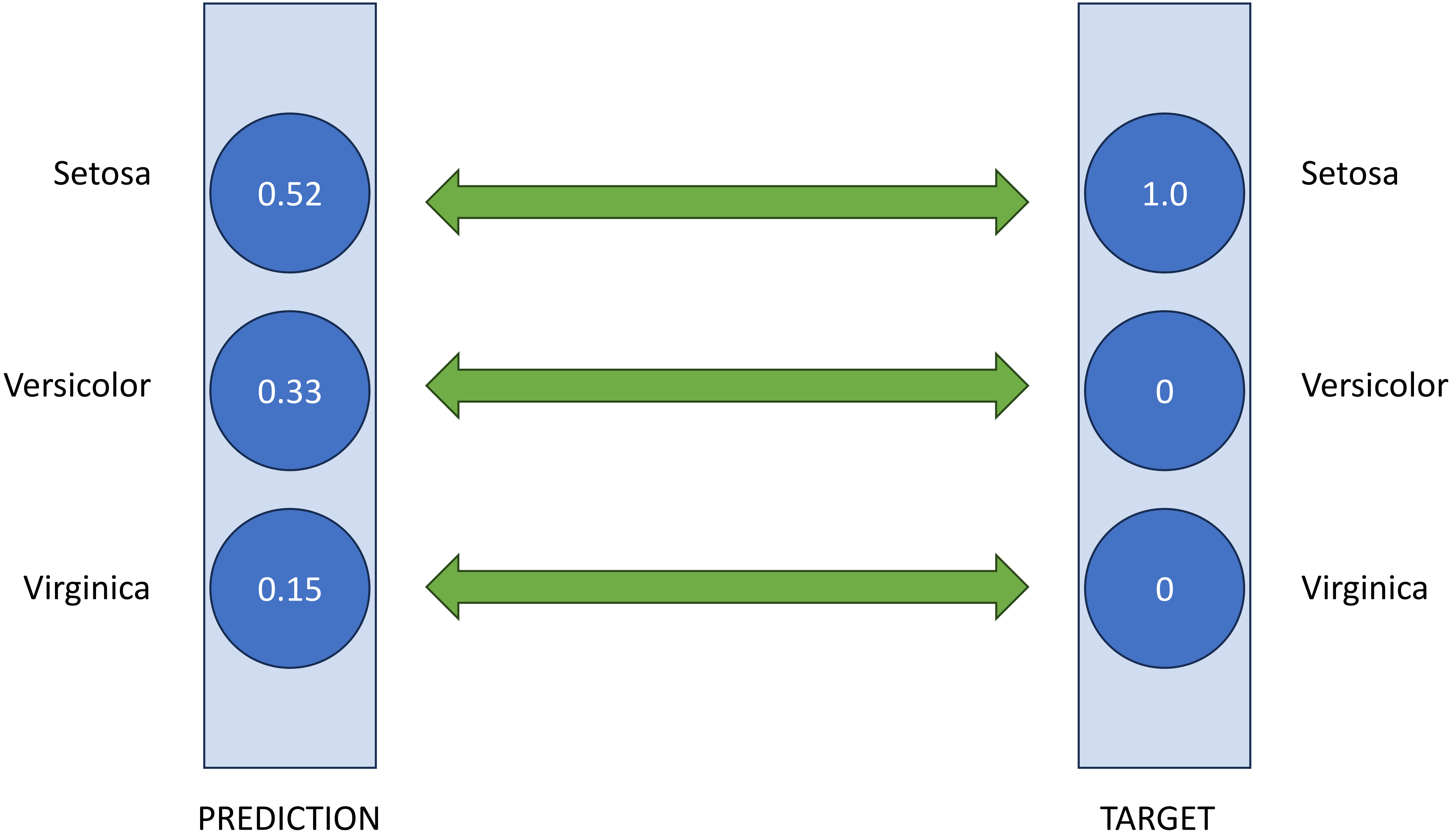

Setosa
0.52
Versicolor
0.33
Virginica
0.15
PREDICTION


Setosa
0.52
Versicolor
0.33
Virginica
0.15
PREDICTION
Setosa
Versicolor
Virginica
TARGET

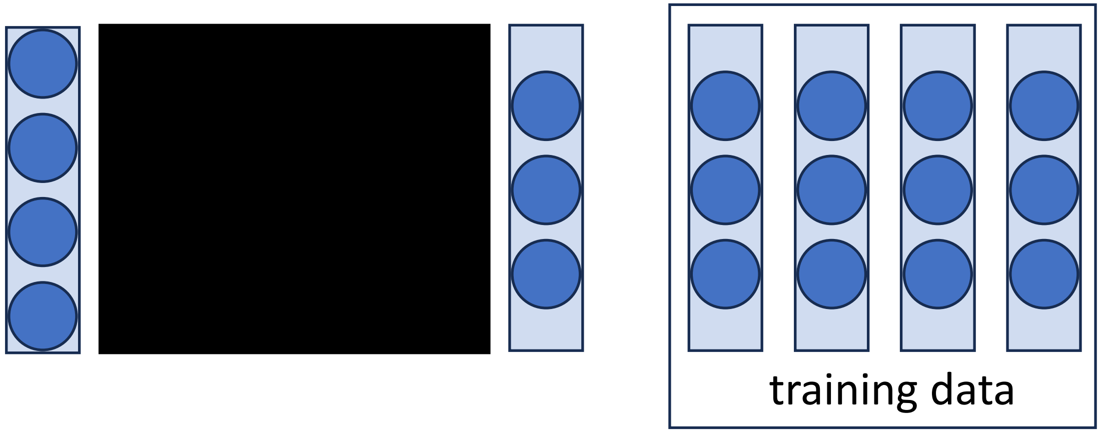

Setosa
Versicolor
Virginica
0.52
0.33
0.15
PREDICTION
1.0
0
0
TARGET
Setosa
Versicolor
Virginica

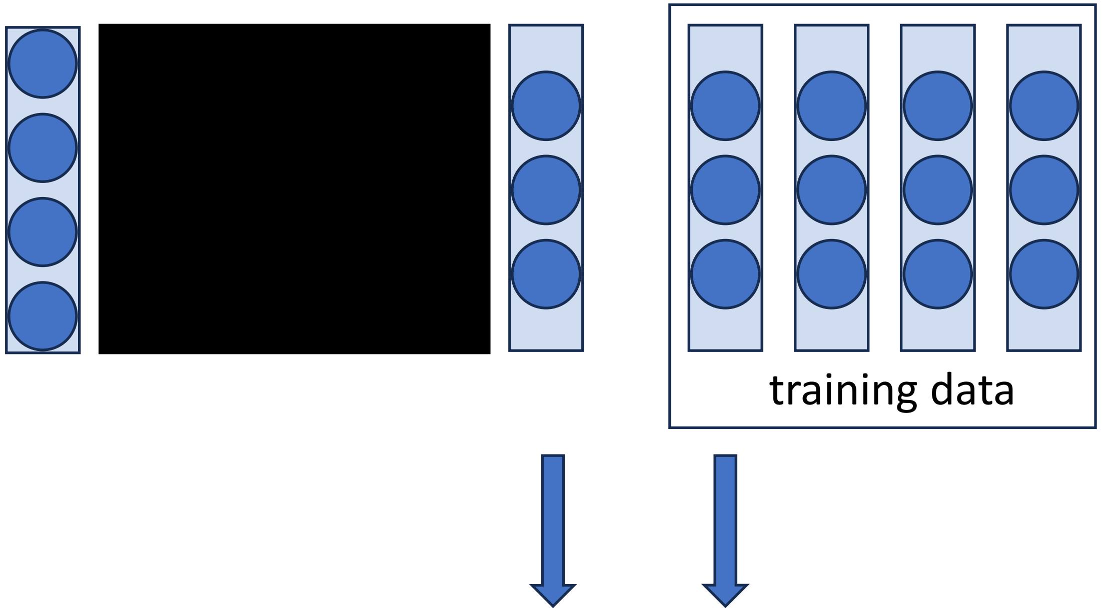

"one hot" encoding
Setosa
Versicolor
Virginica
0.52
0.33
0.15
PREDICTION
1.0
0
0
Setosa
Versicolor
Virginica
TARGET


Setosa
0.52
1.0
Setosa
Versicolor
0.33
0
Versicolor
Virginica
0.15
0
Virginica
PREDICTION
TARGET

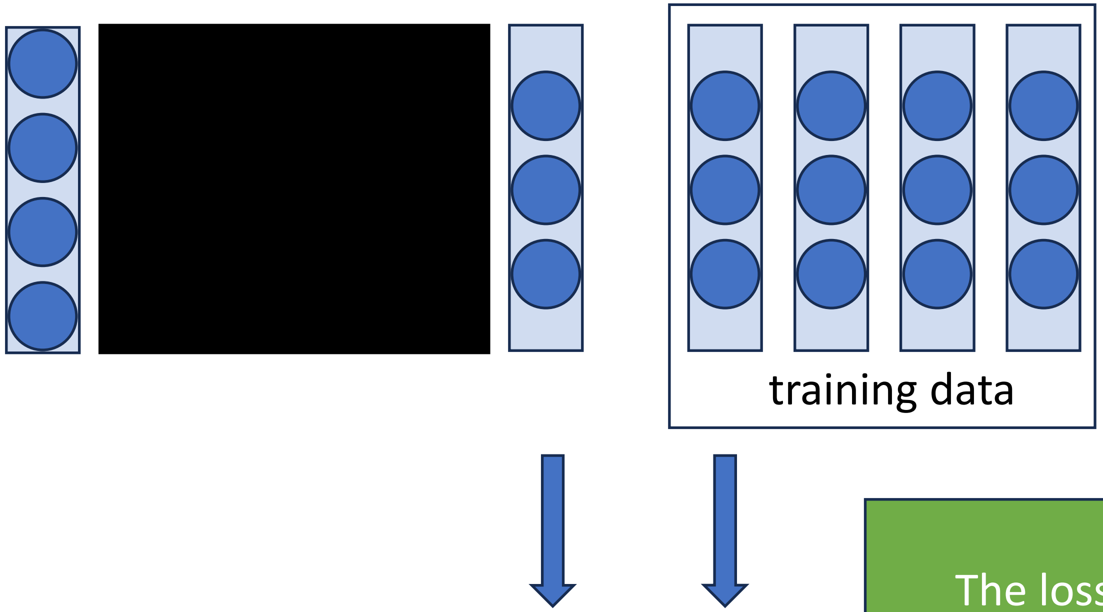


We can do better!


training data

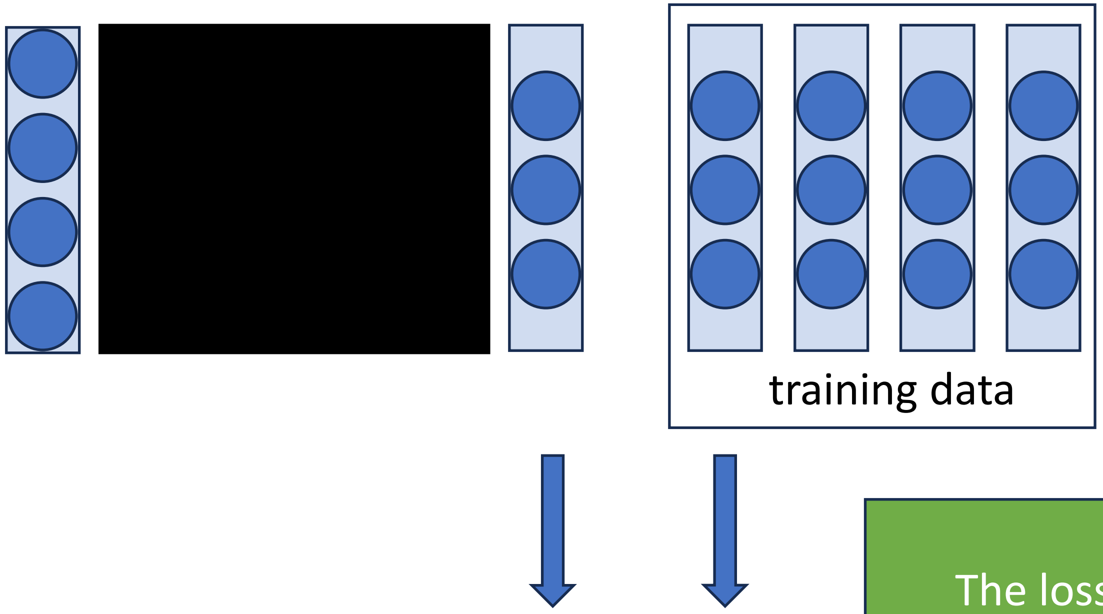


loss_function( )



loss_function( )

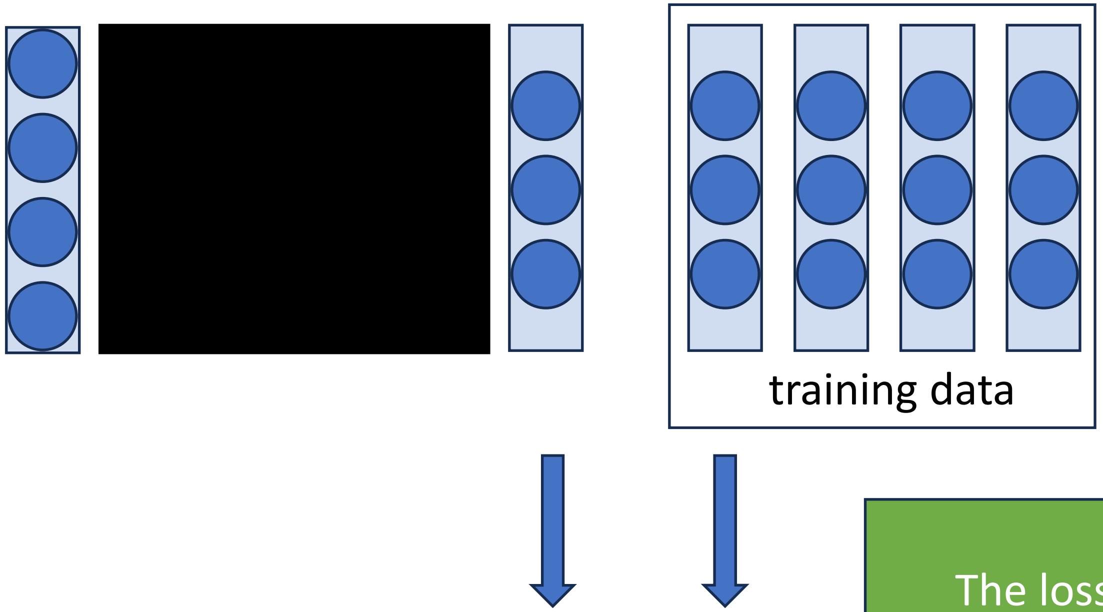
The loss (or error) function is a measure of similarity.



loss_function(   )

The loss (or error) function is a measure of similarity.

It tells us how good the prediction is.

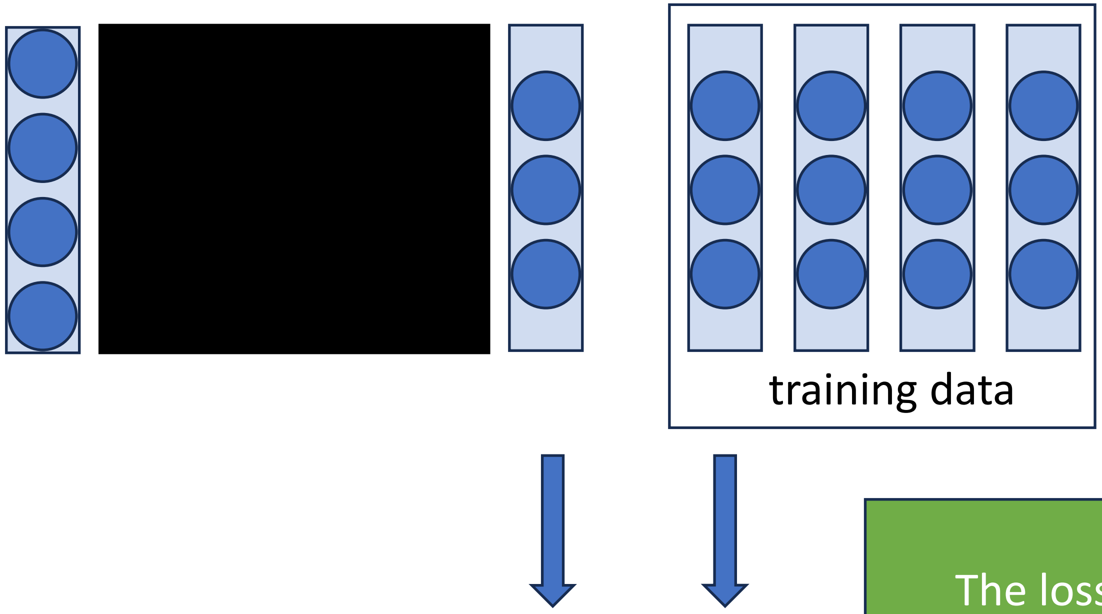


loss_function( )

The loss (or error) function is a measure of similarity.

The goal is to minimize the loss / error.

It tells us how good the prediction is.

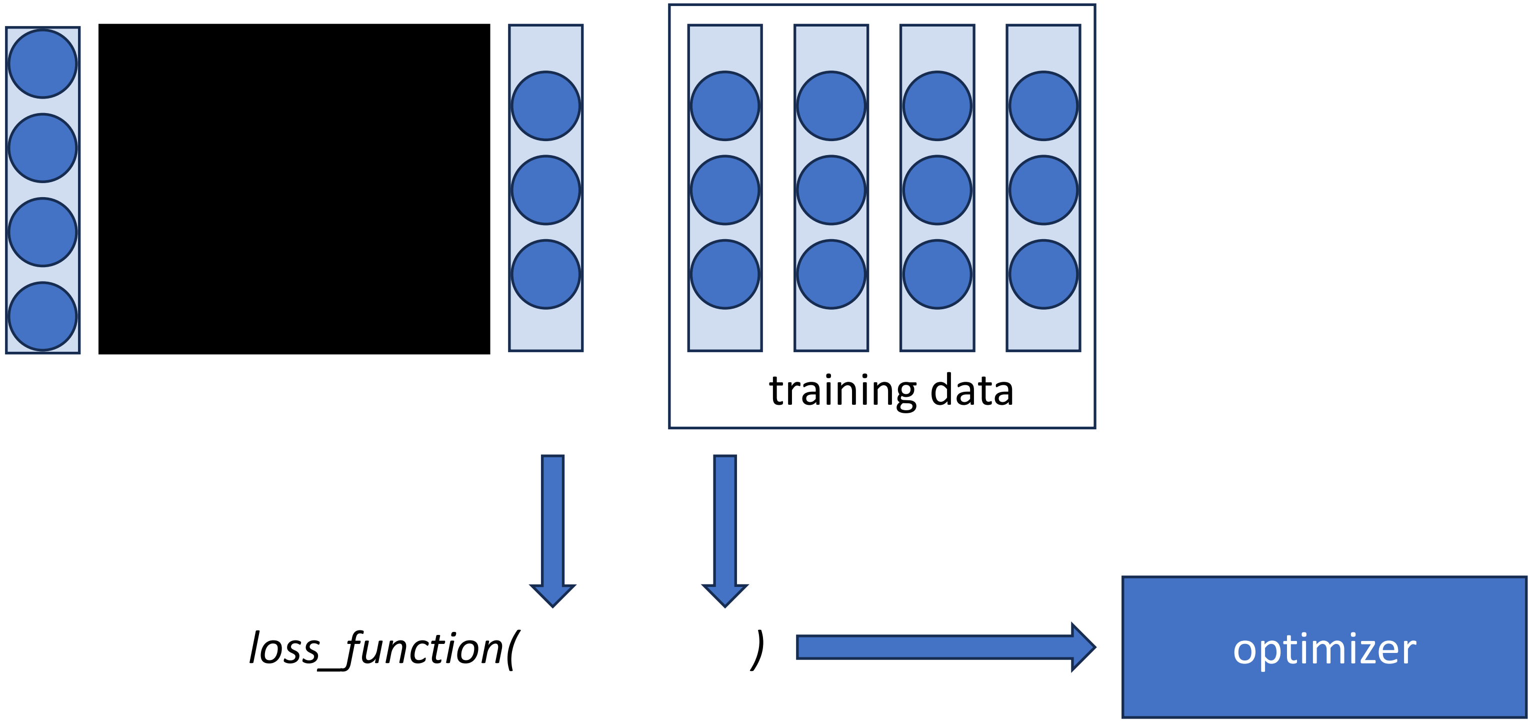


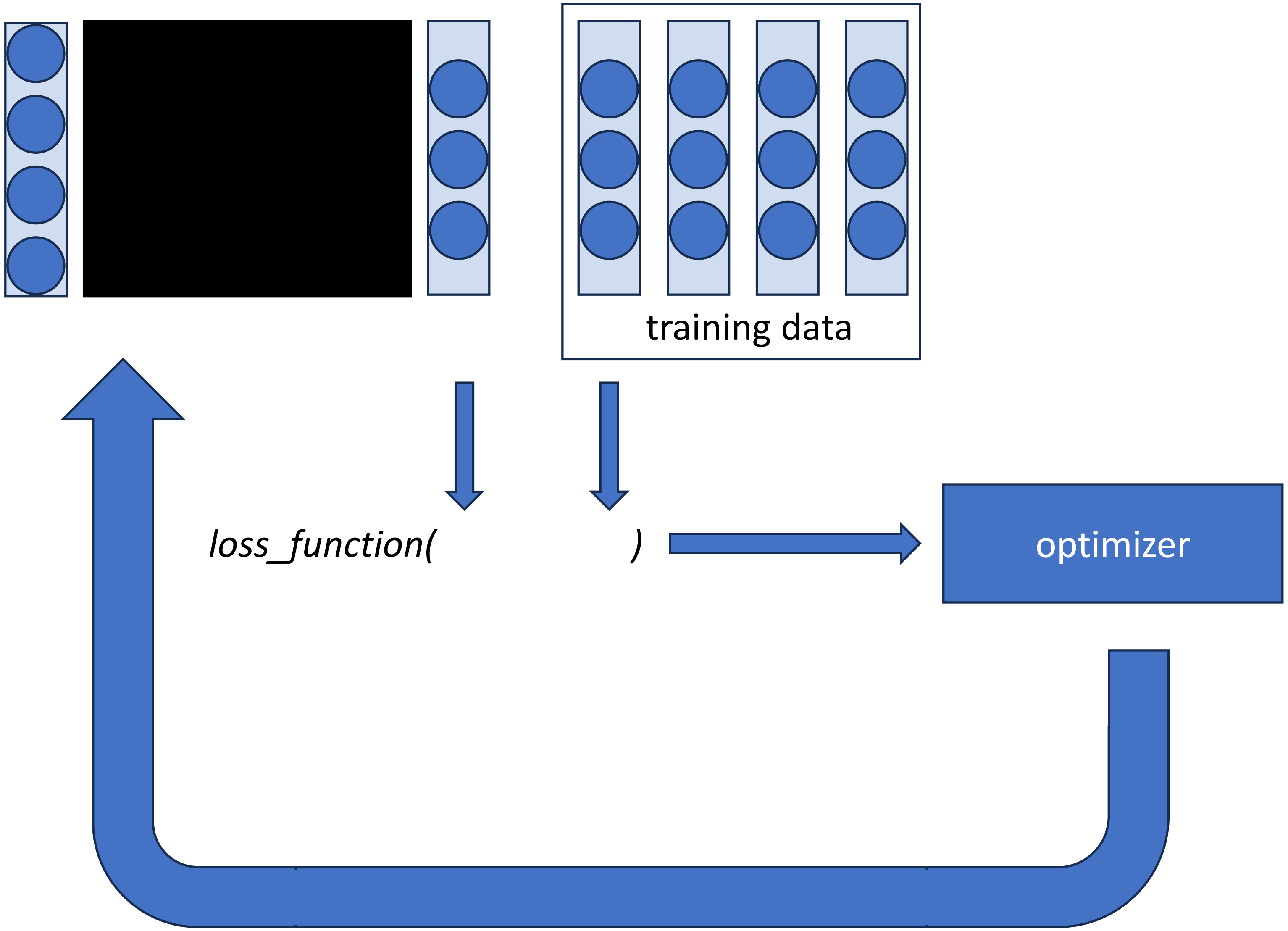
loss_function( )

The loss (or error) function is a measure of similarity.

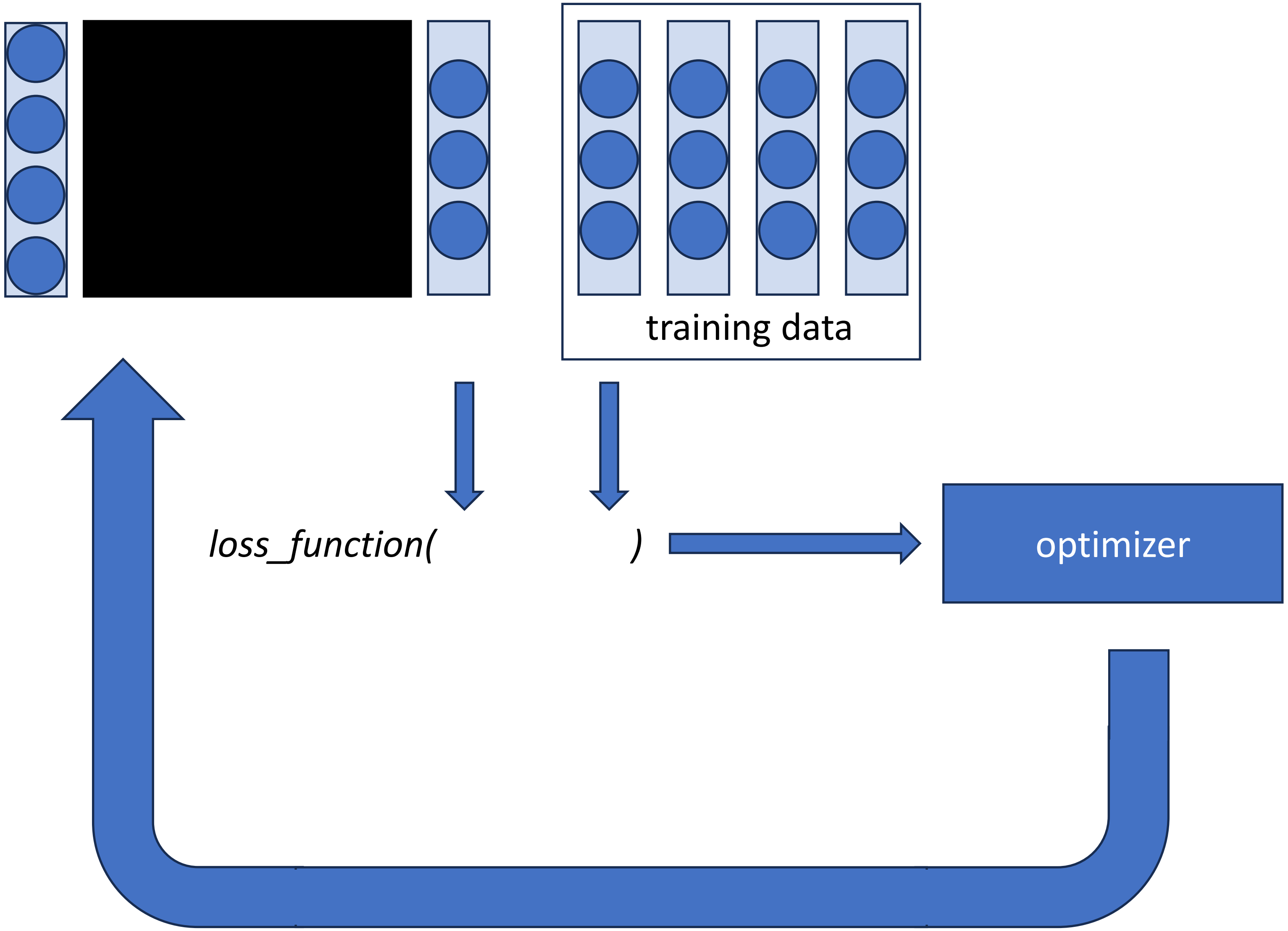
The goal is to minimize the loss / error.

It tells us how good the prediction is.

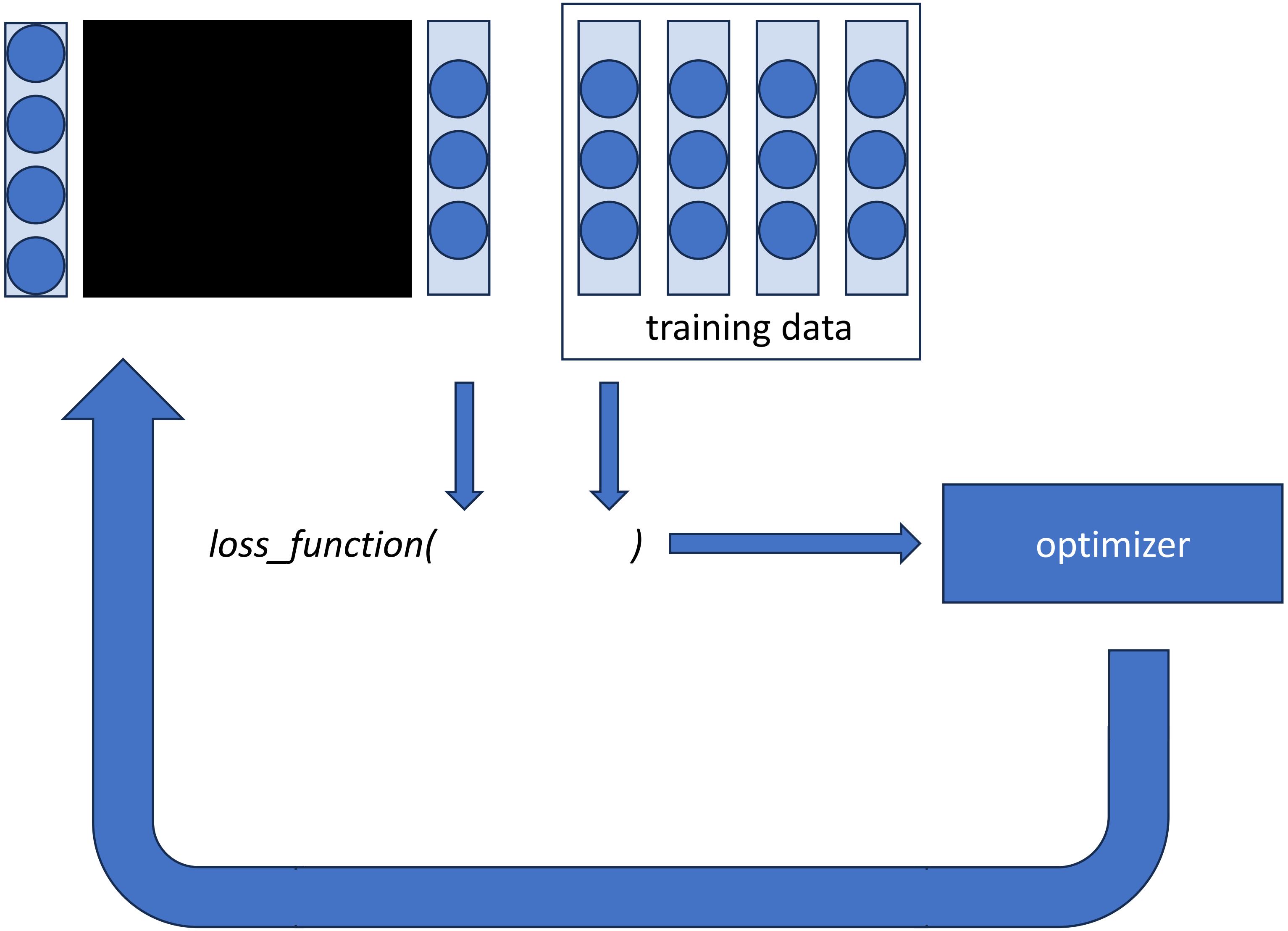

training data
loss_function(   )
optimizer



The optimizer determines how to adjust the weights in the network.

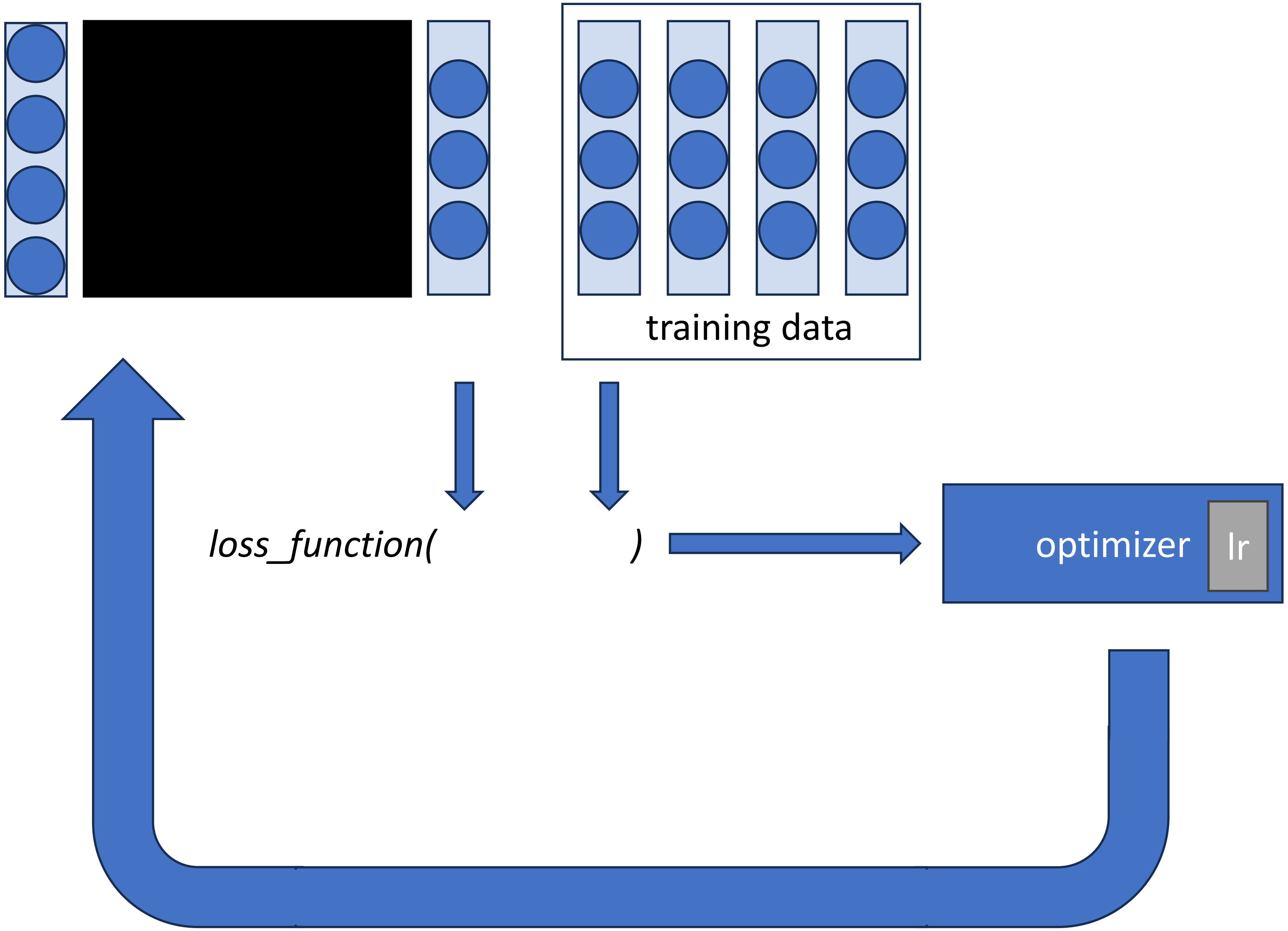


In addition to the weight parameters, the networks can have “hyperparameters”.

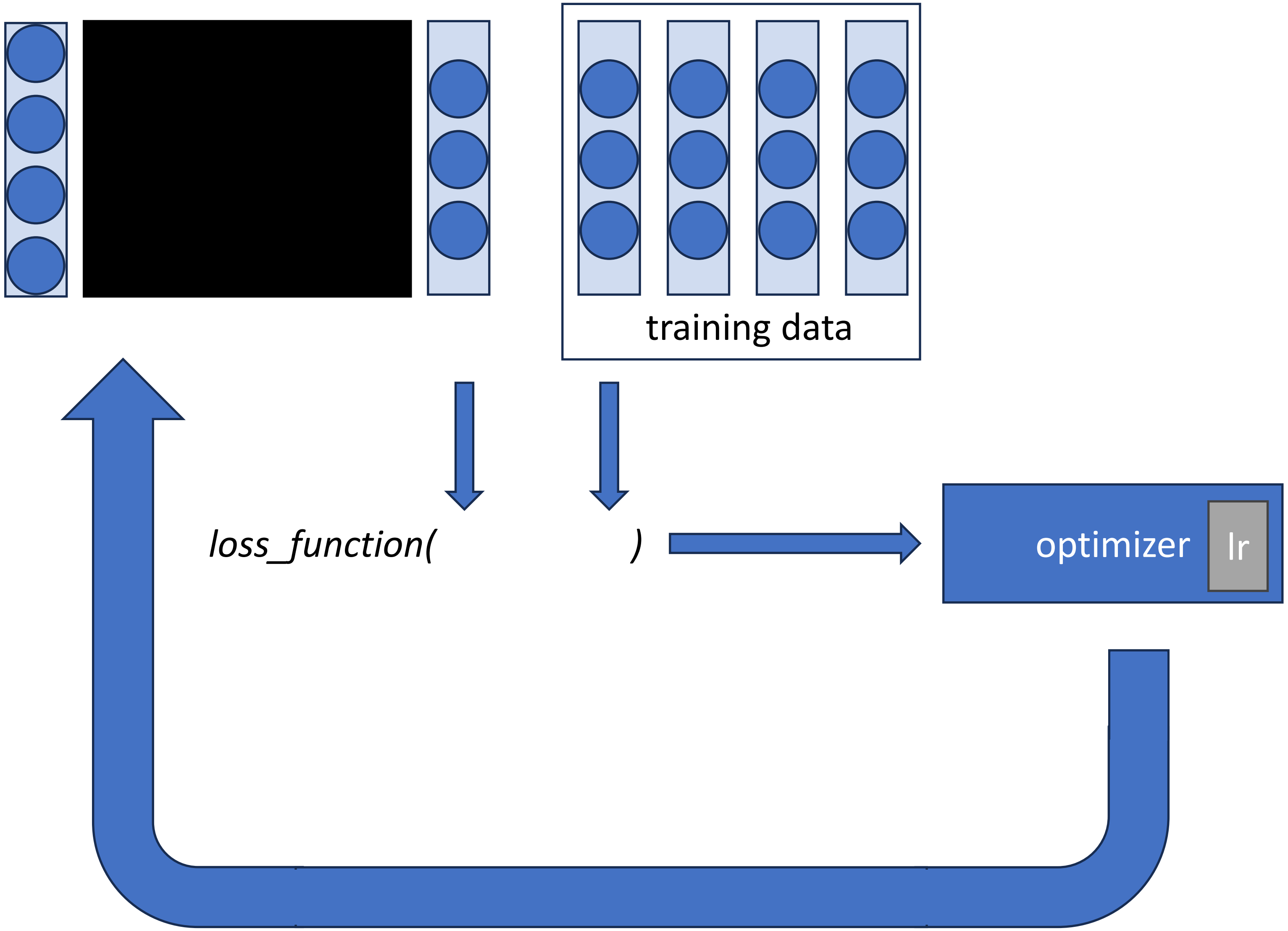


In addition to the weight parameters, the networks can have “hyperparameters”.

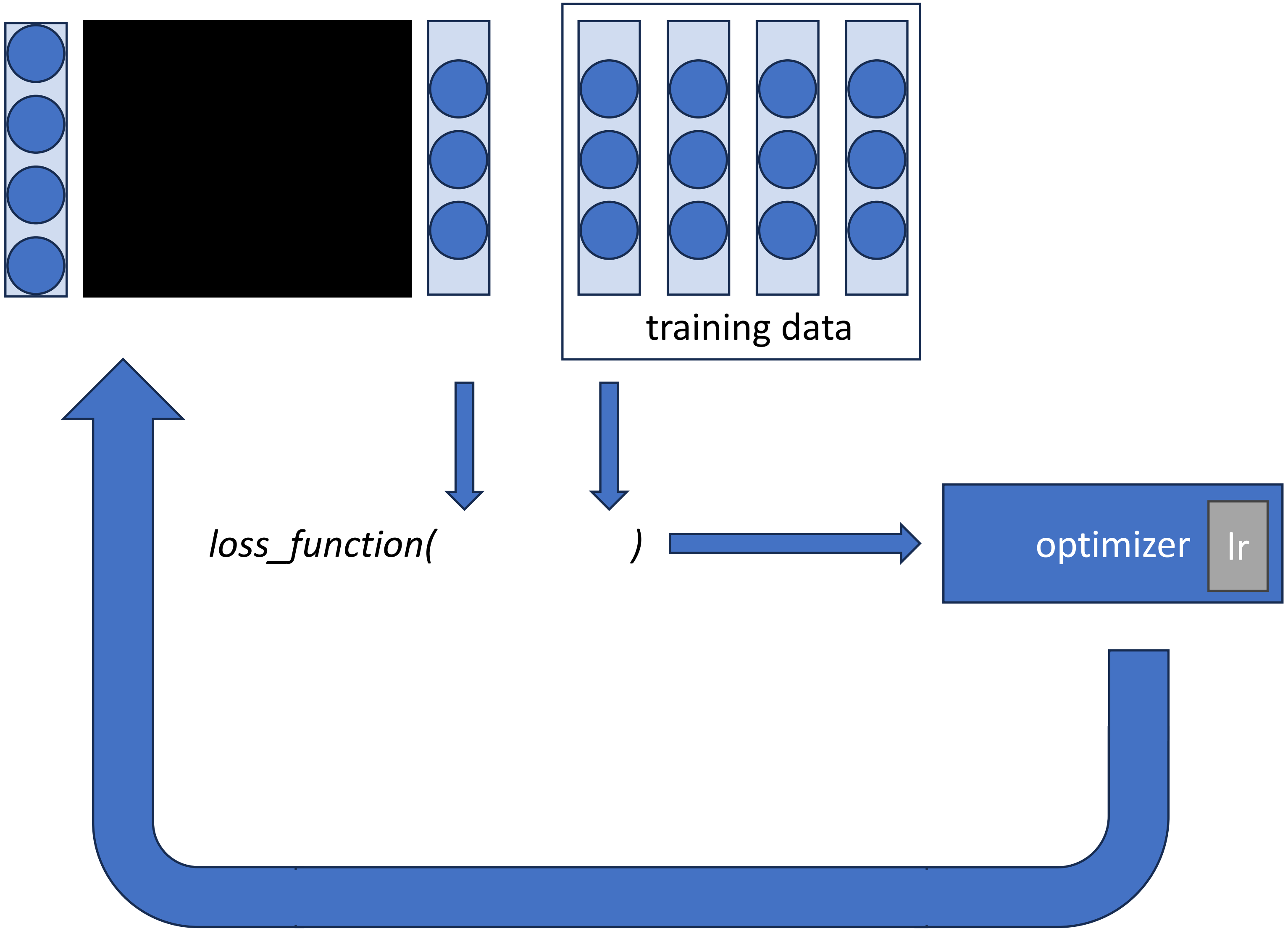
These are parameters that are not updated during training.



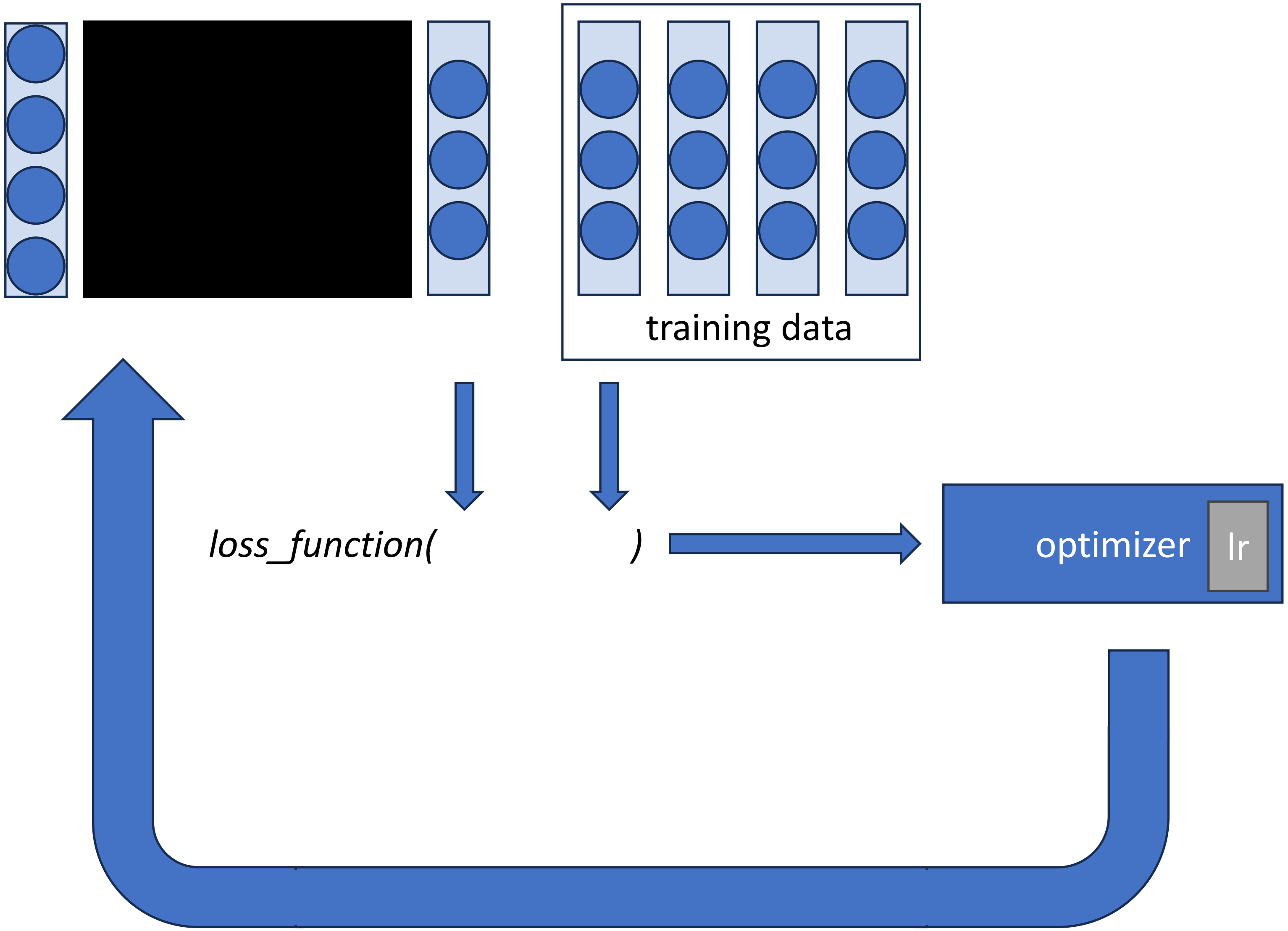
The “learning rate” is an example of a hyperparameter.



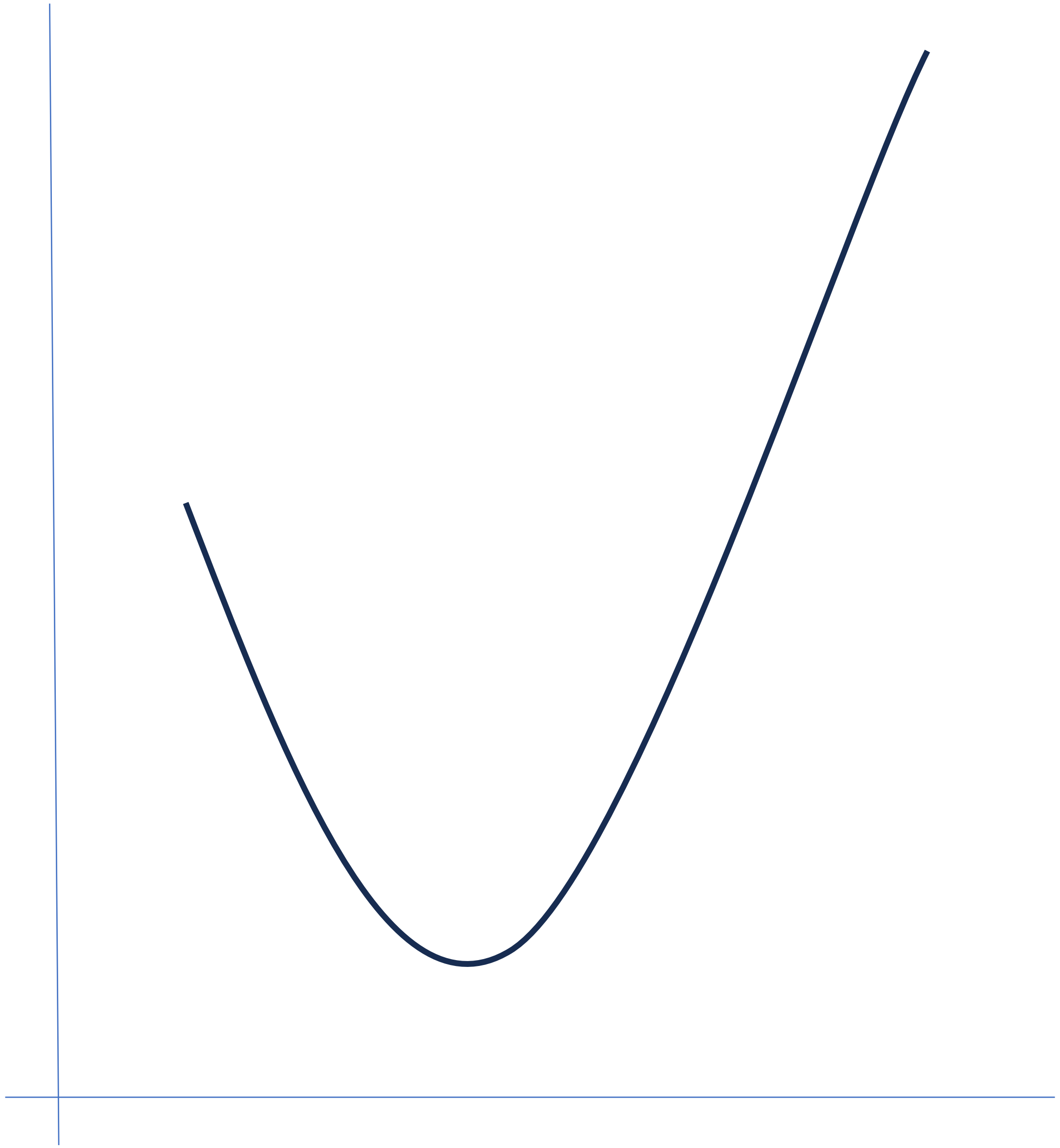
It specifies how fine-tuned the updates to the weight will be.

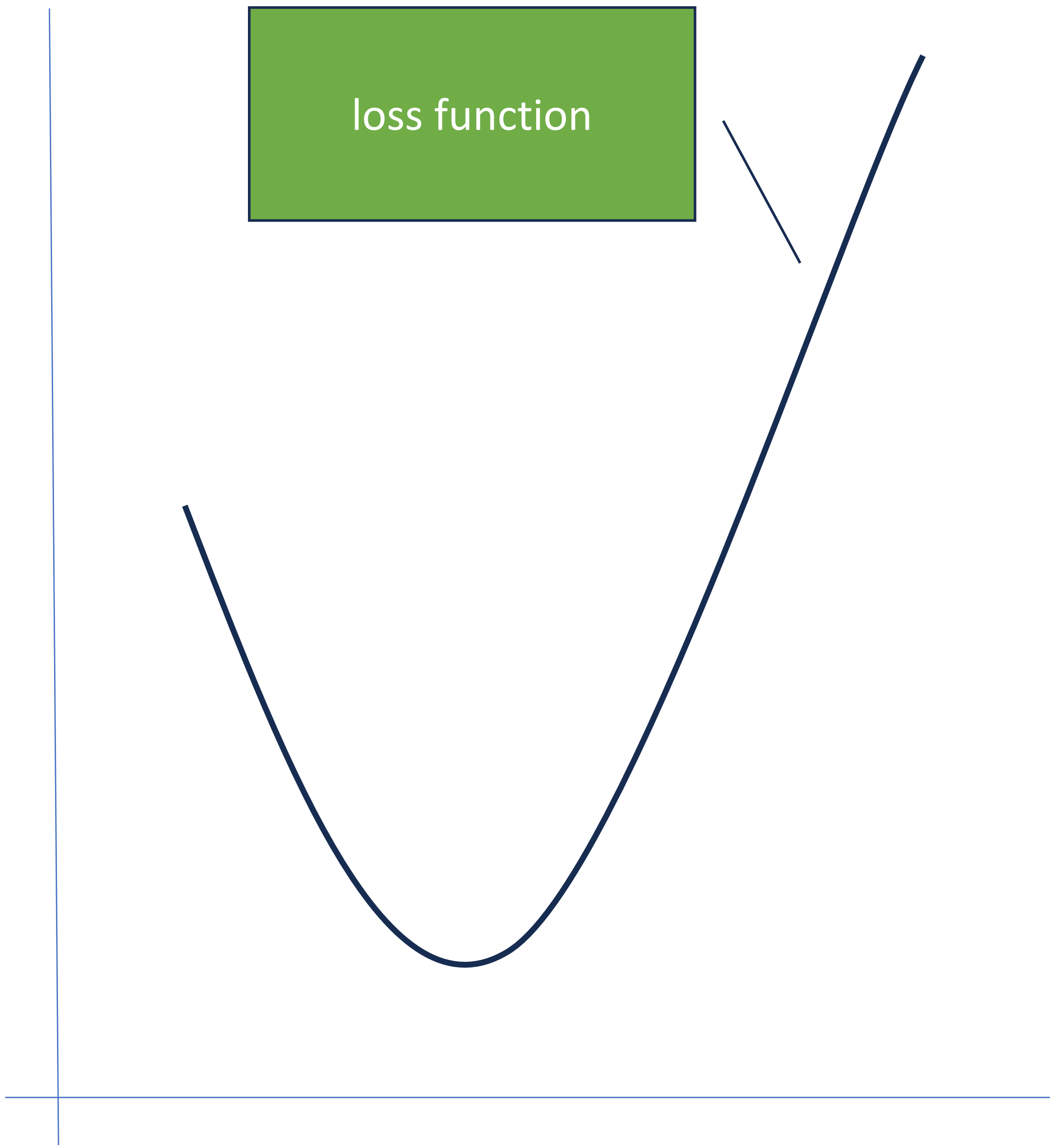


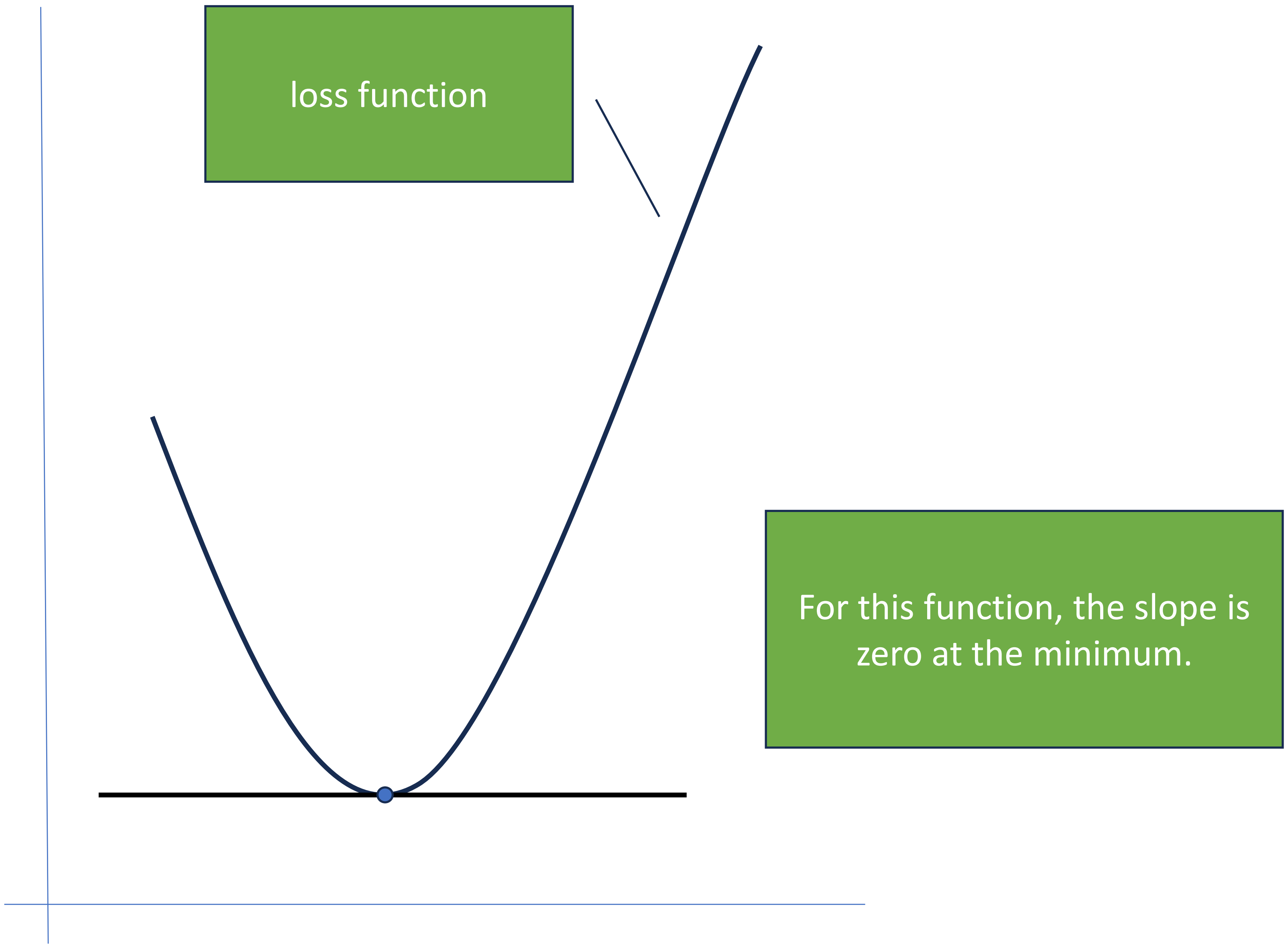
Set it too large, and the optimizer will miss the minimum error.

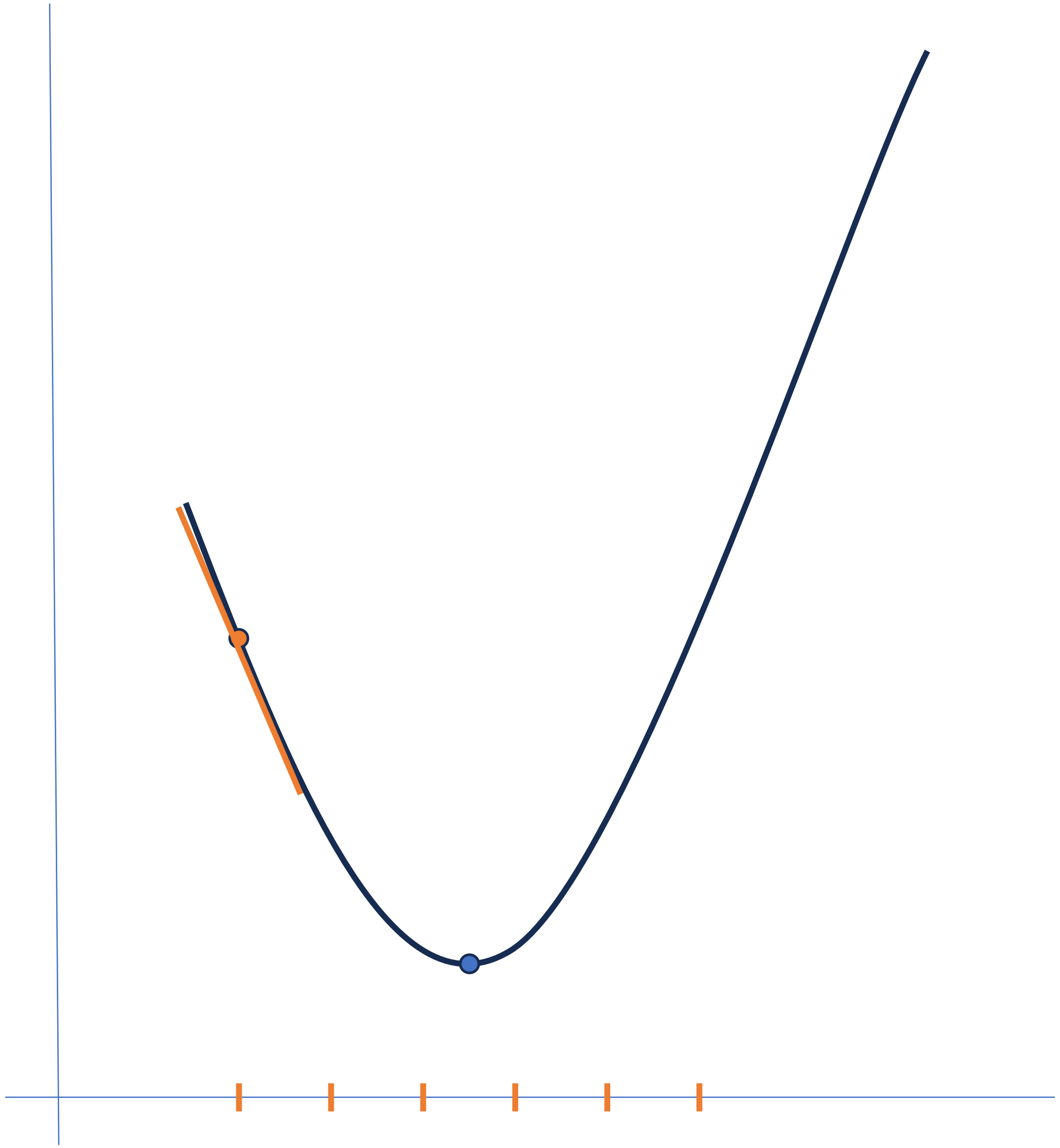


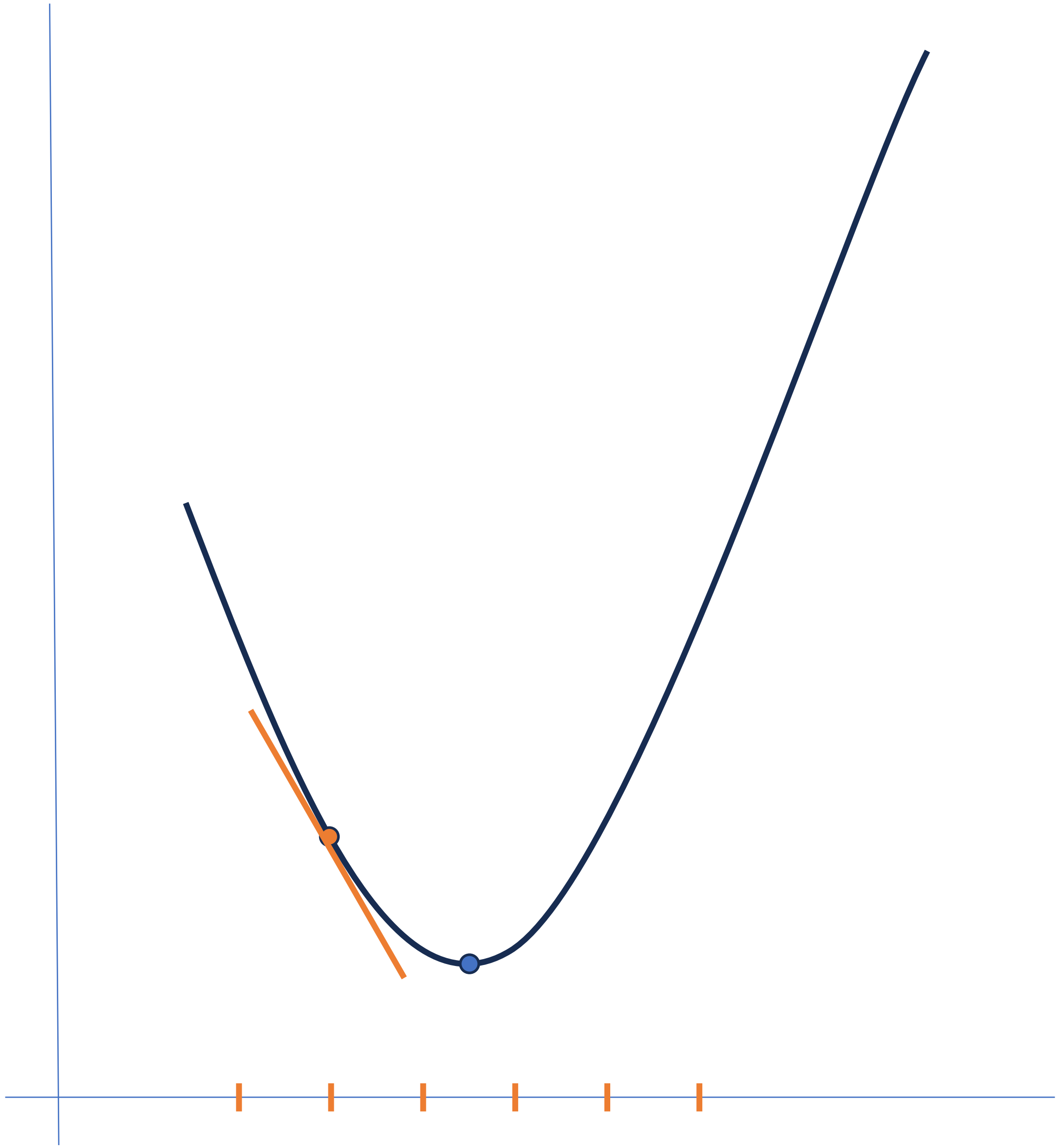
Set it too small, and the optimizer will waste a lot of time.

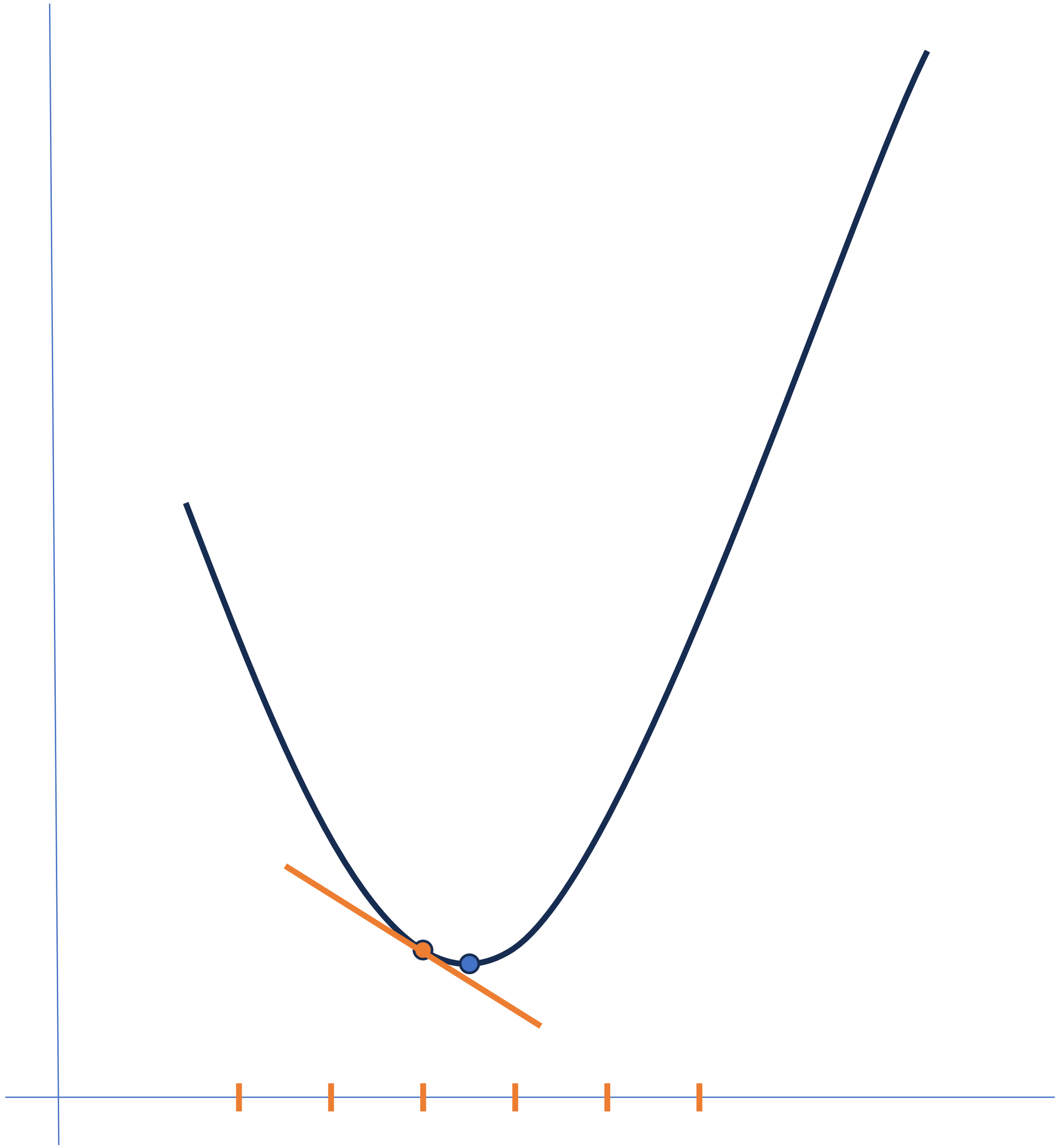


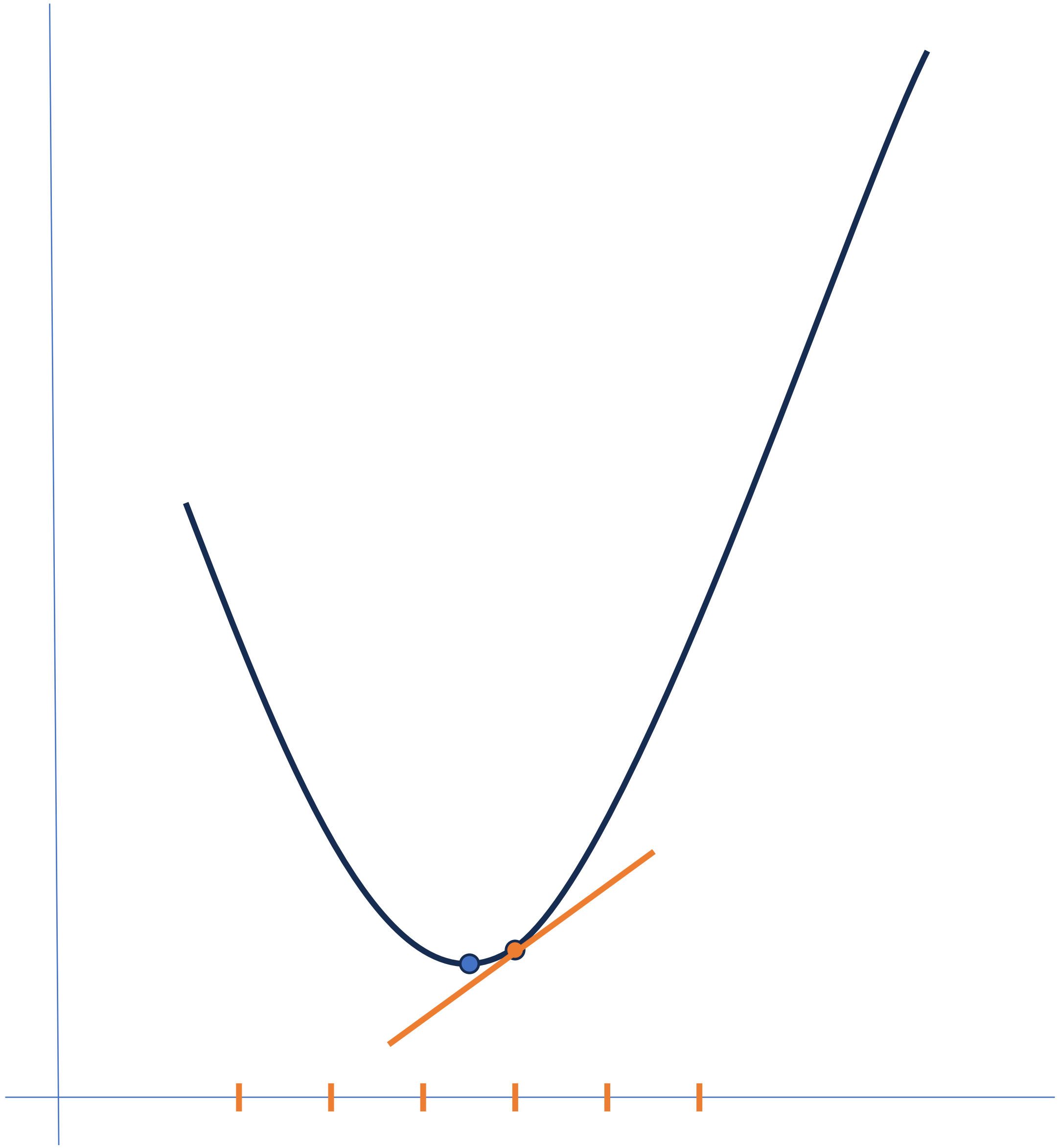

loss function

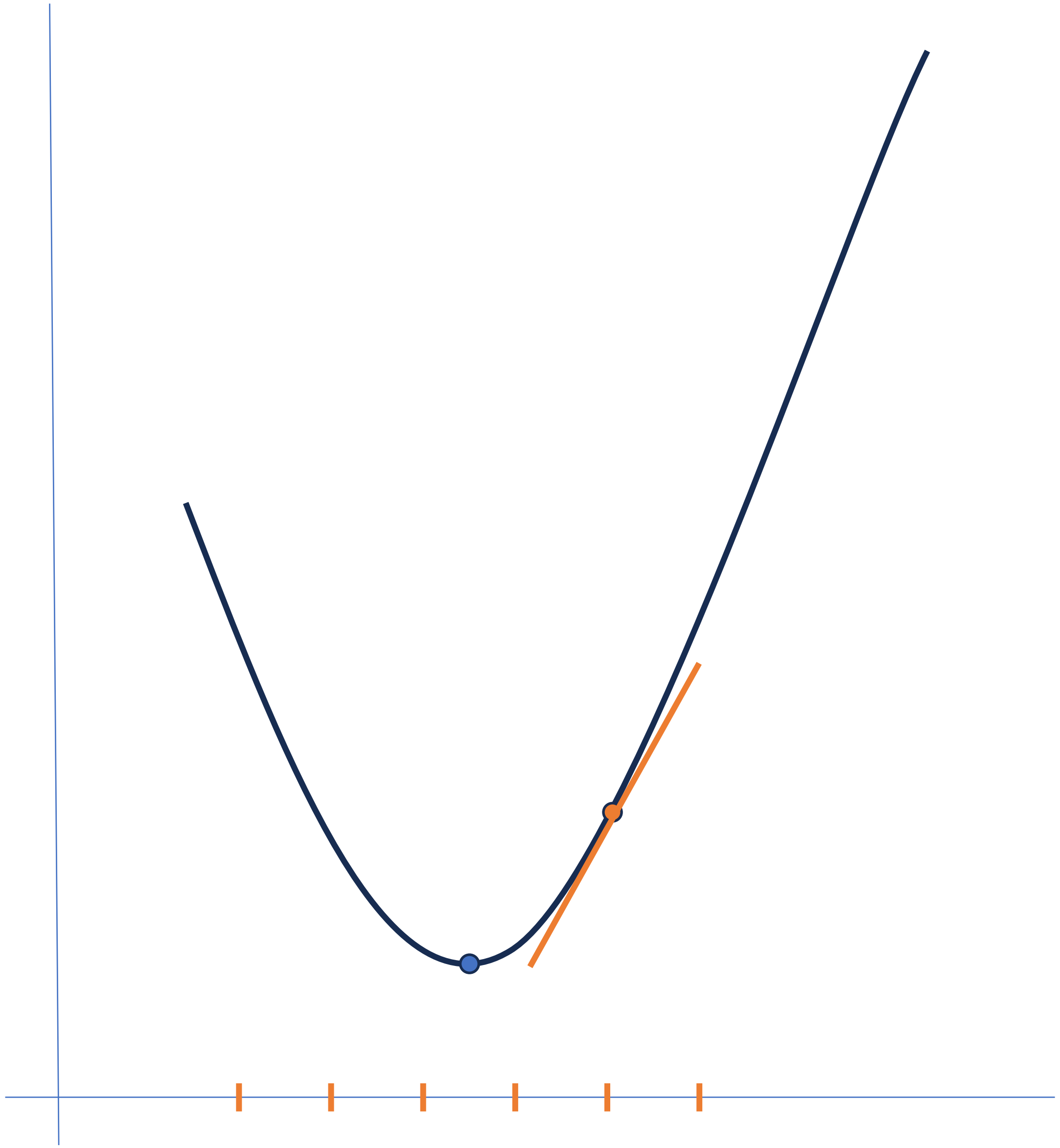

loss function
For this function, the slope is zero at the minimum.

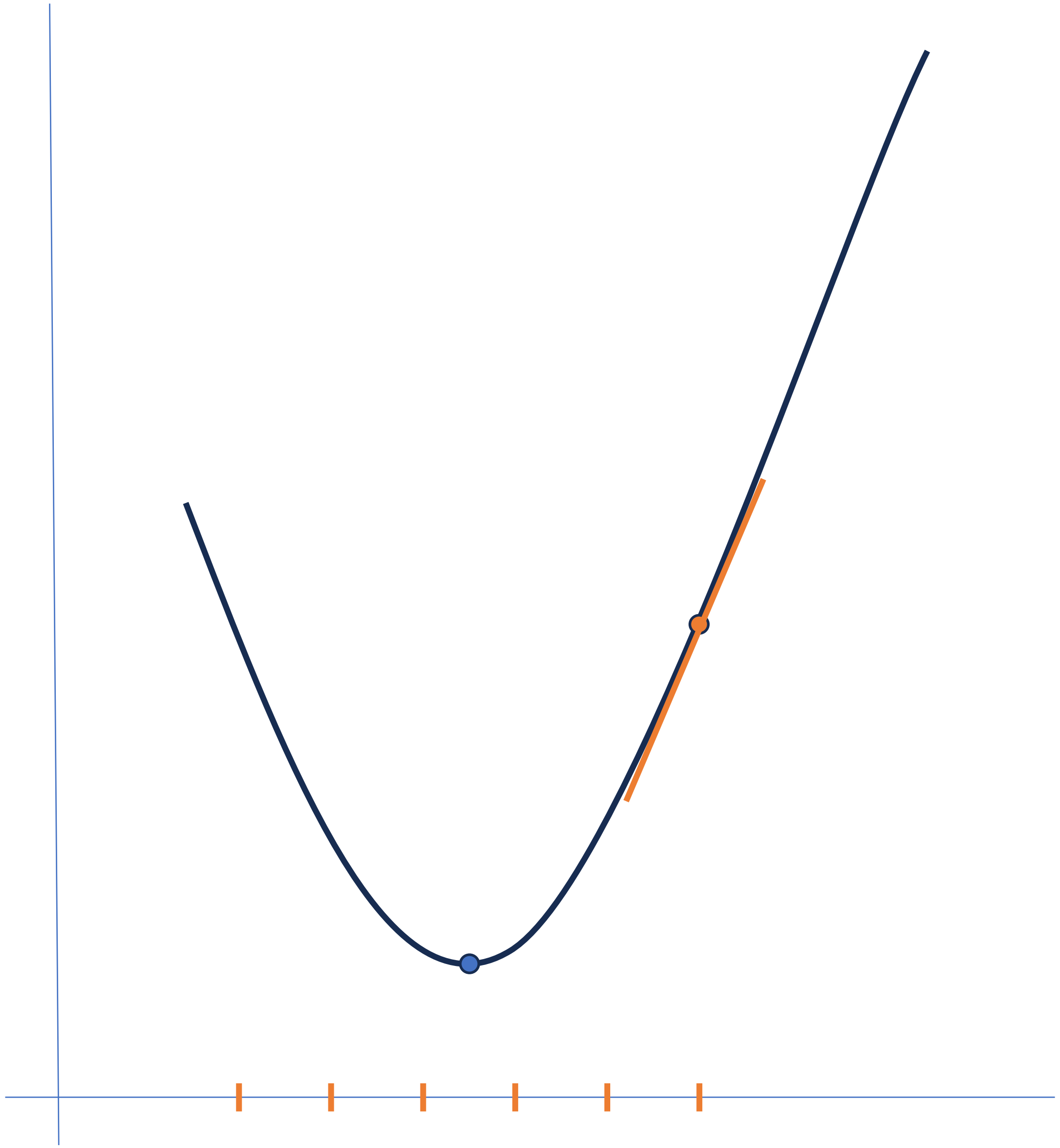


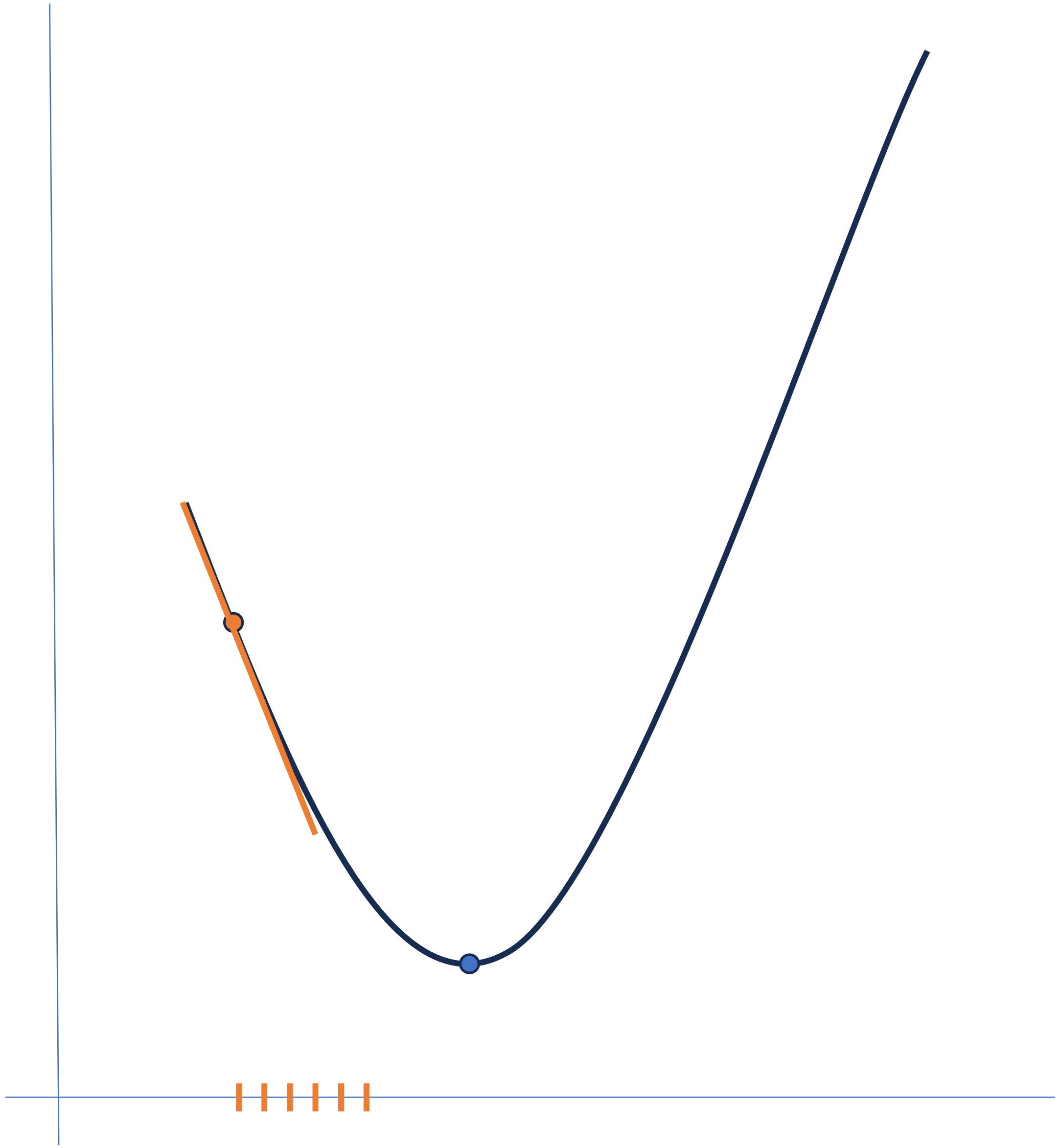


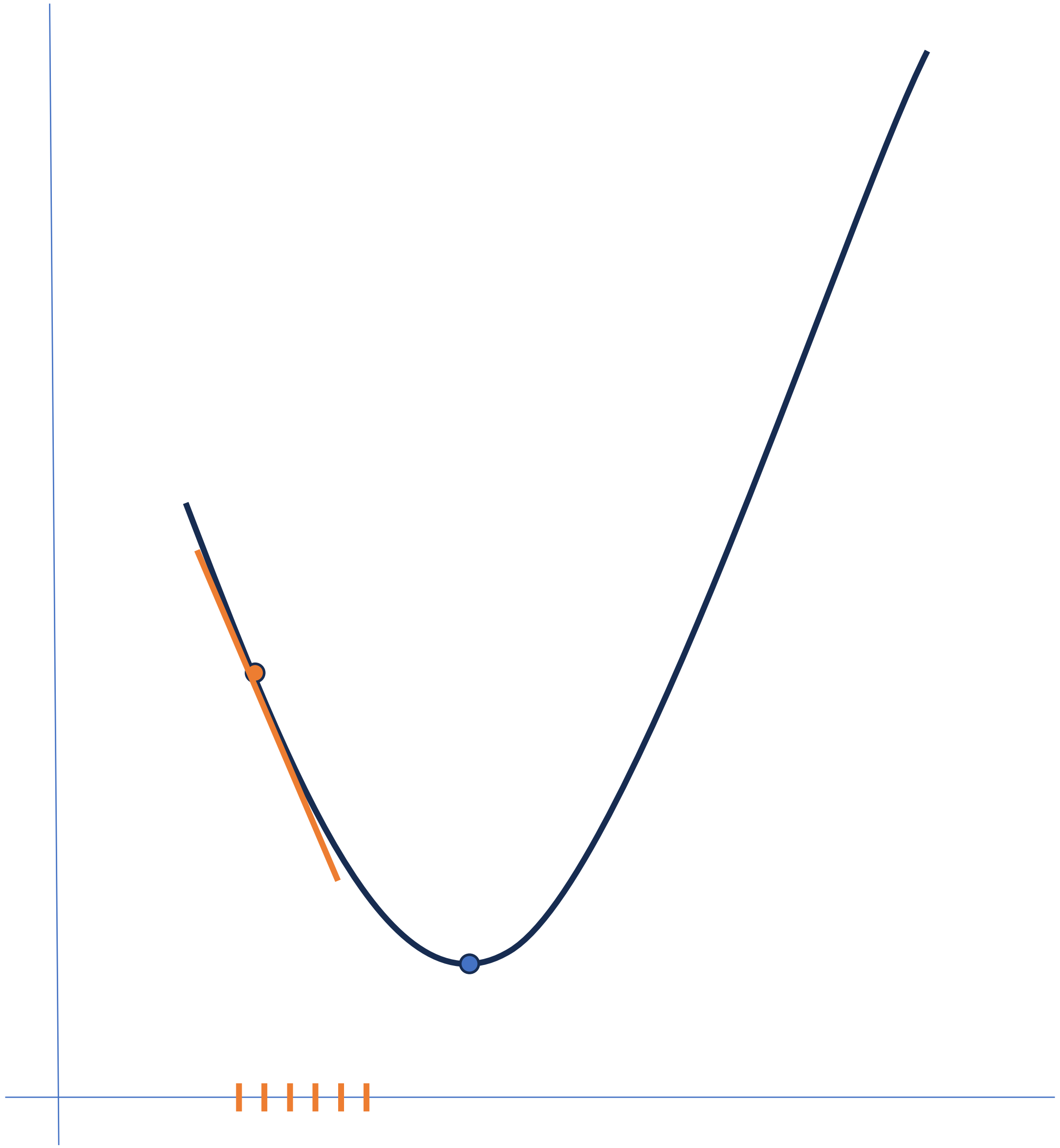


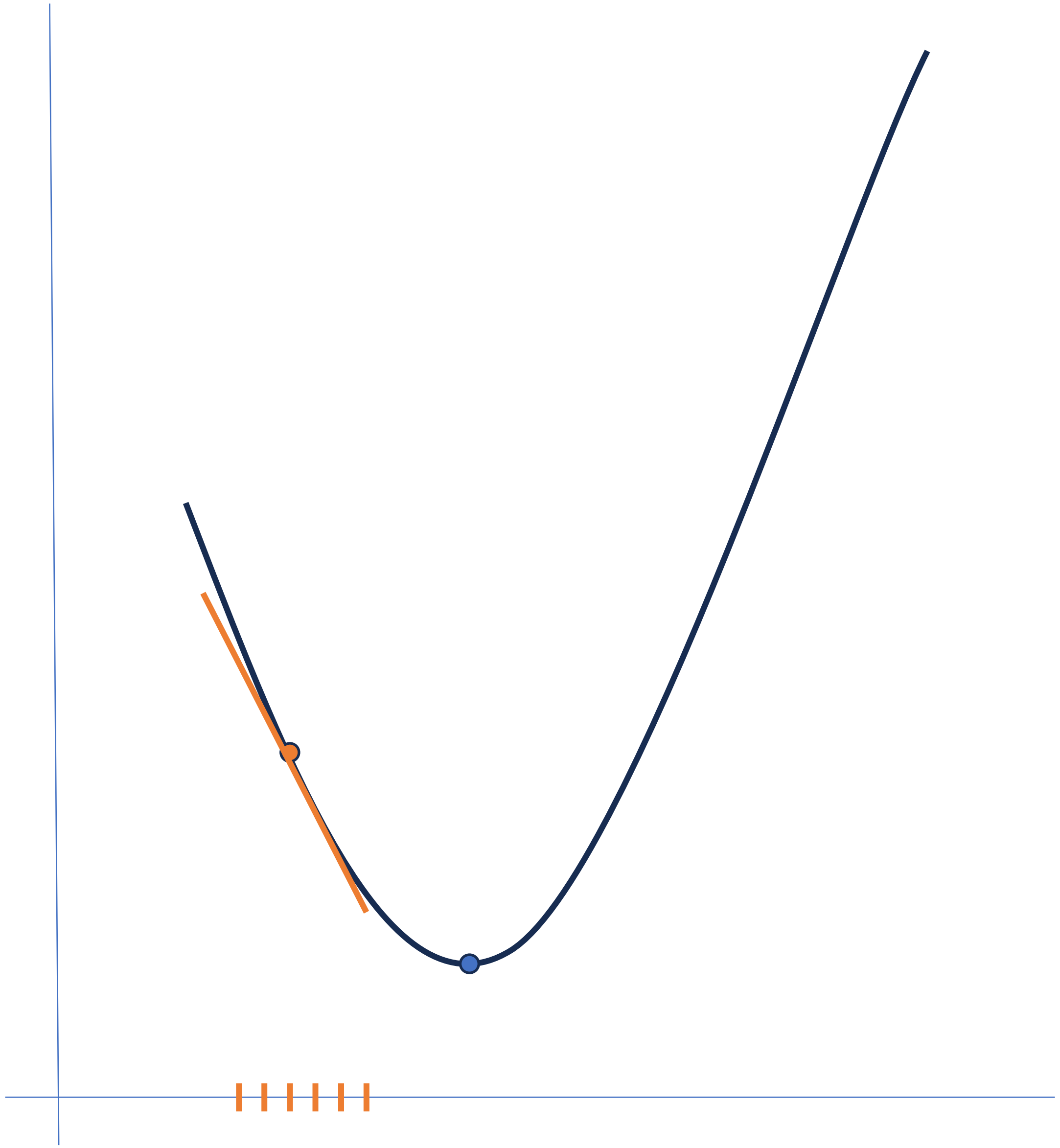


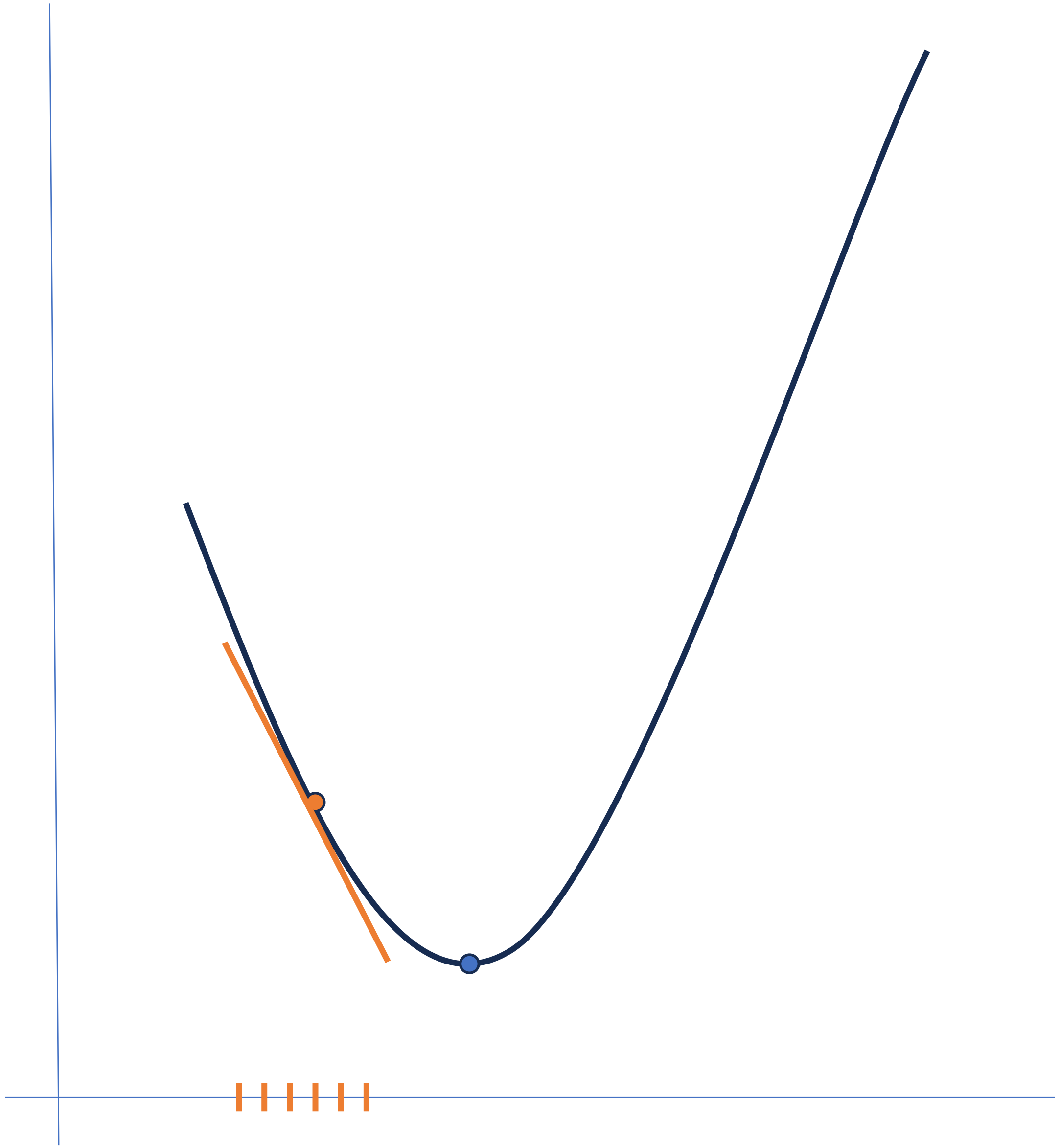


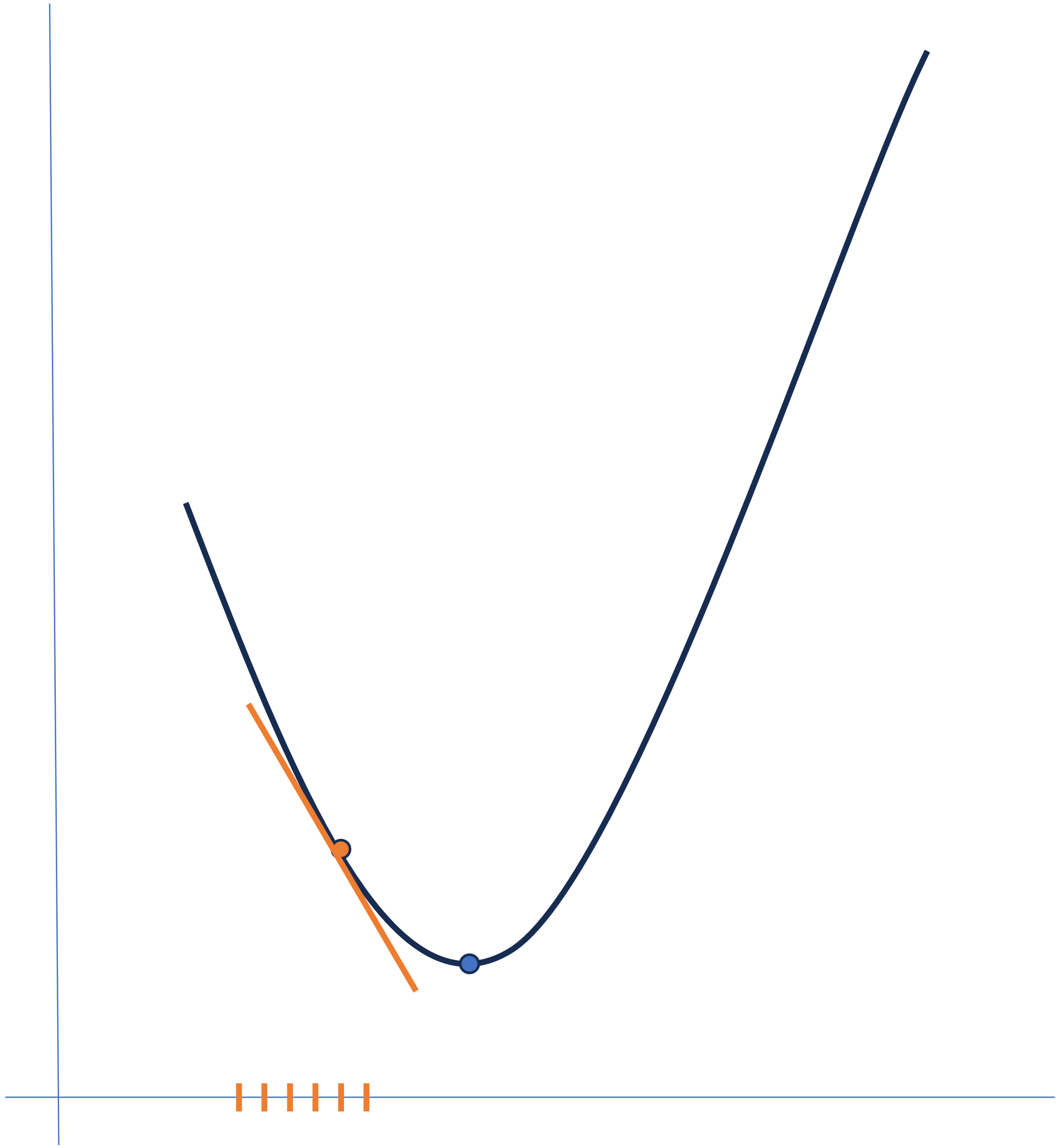


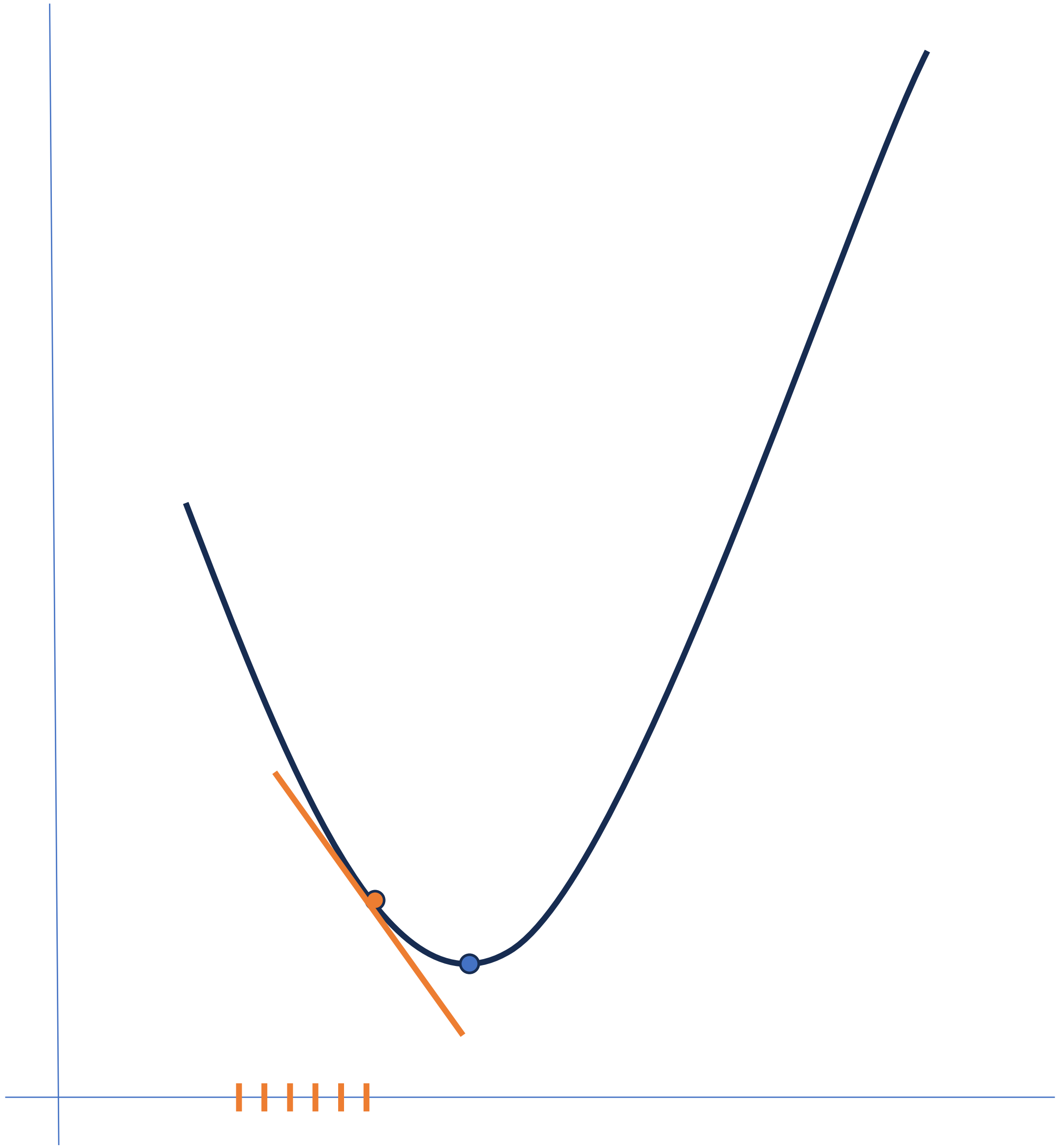


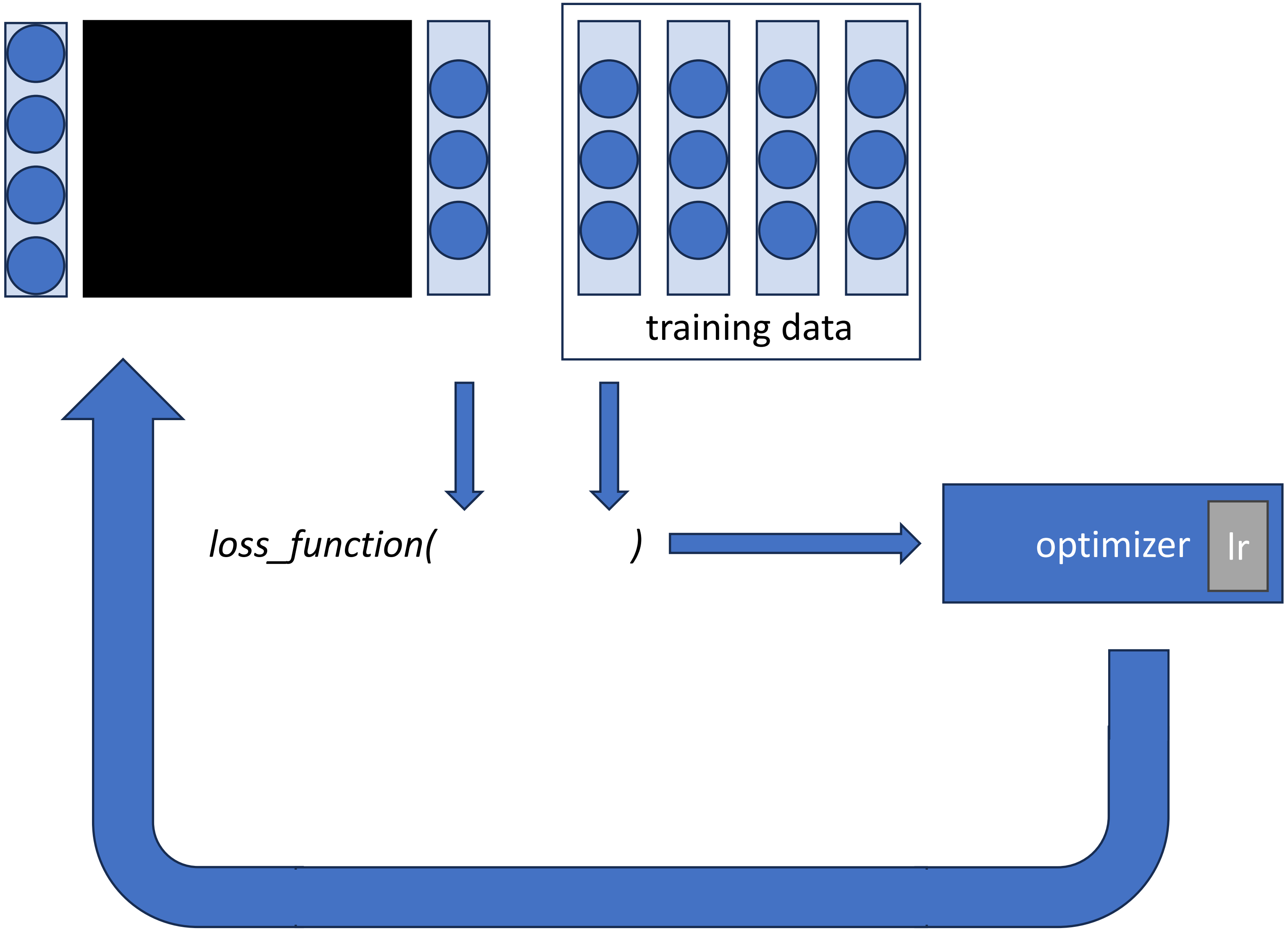




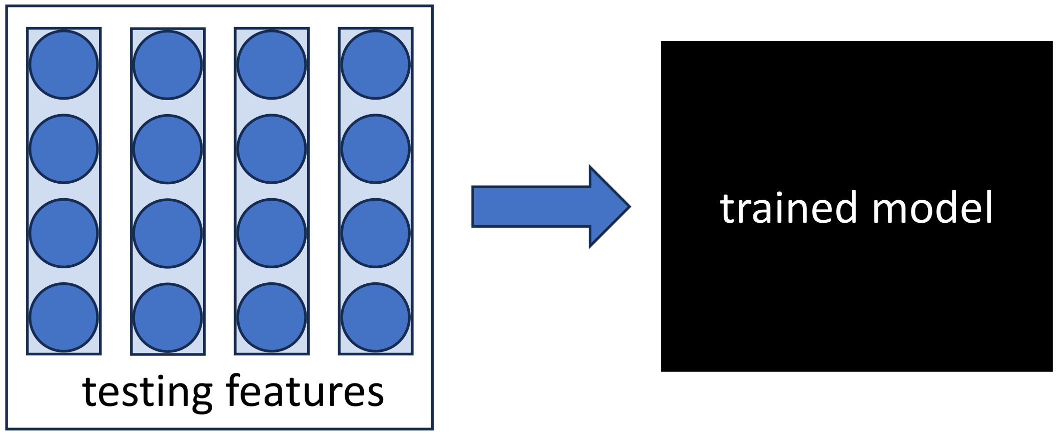


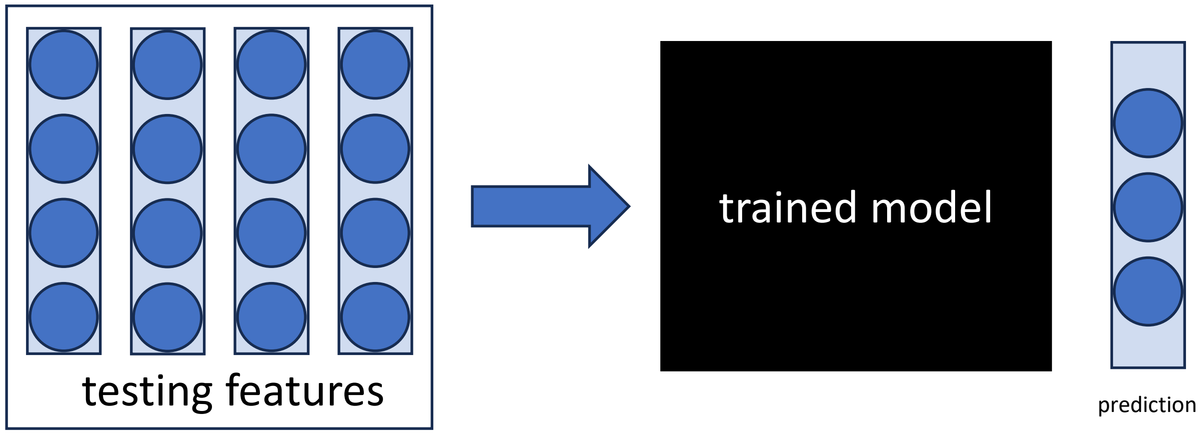




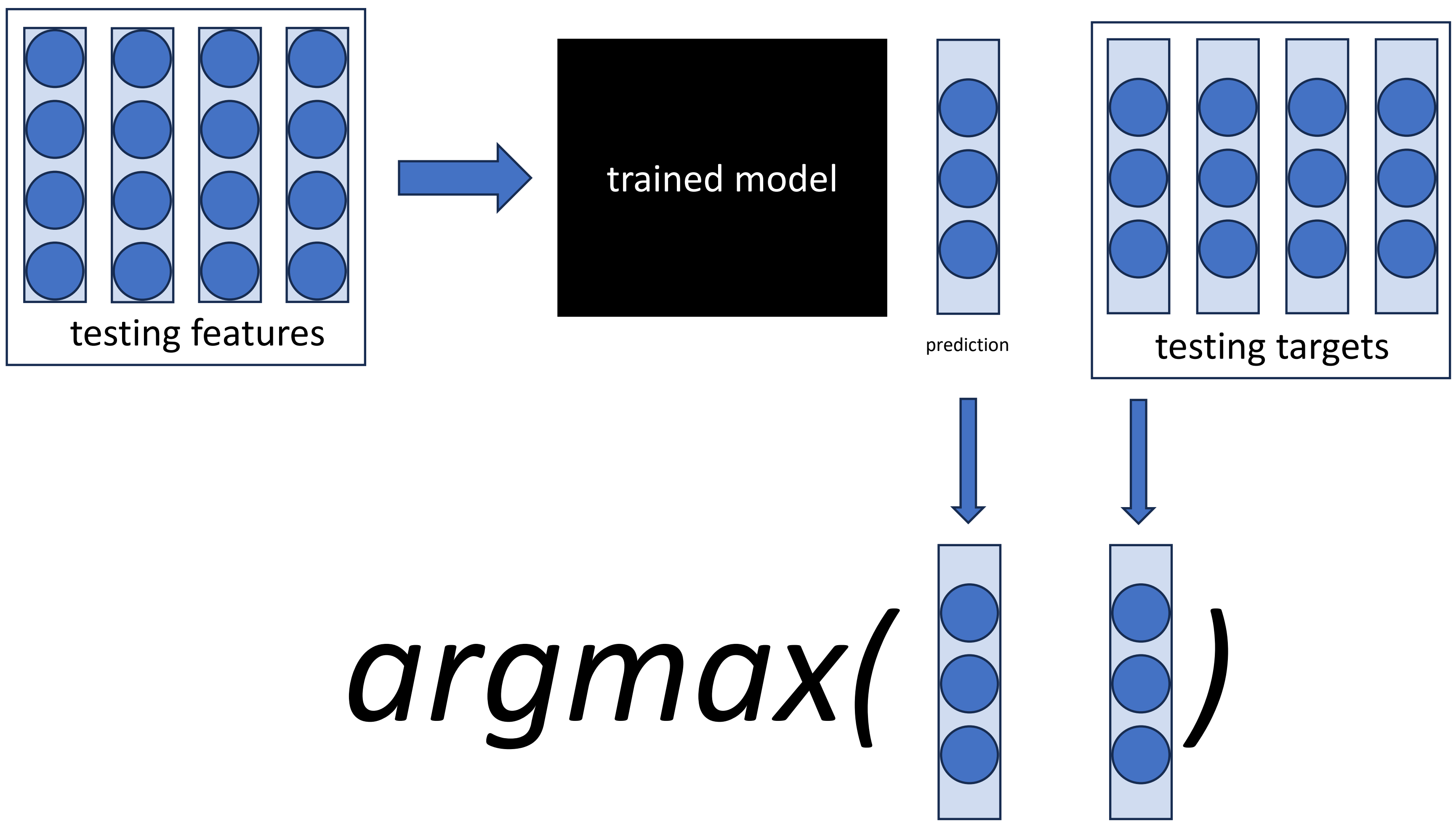


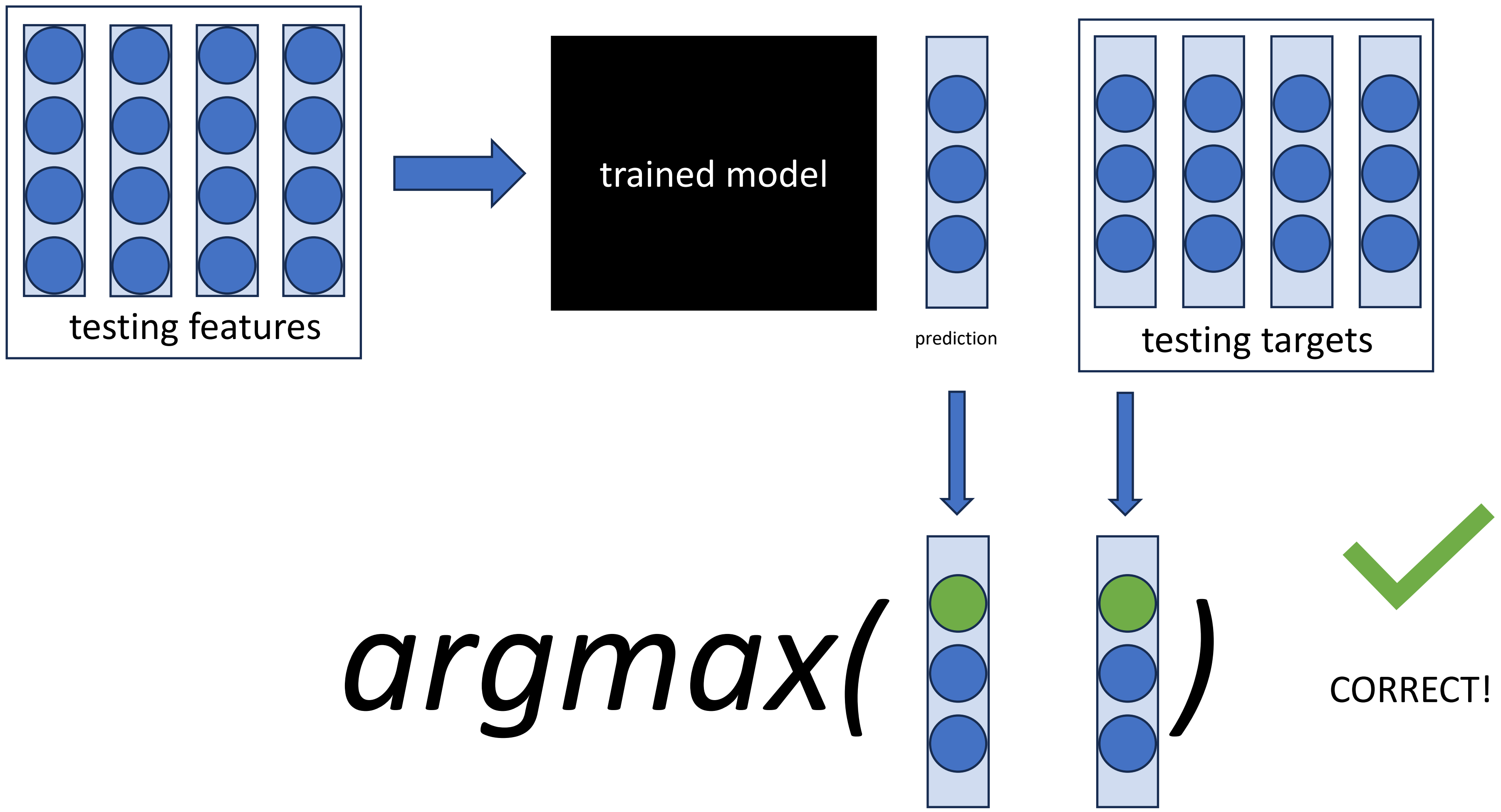
This process is called “training” the model trained model

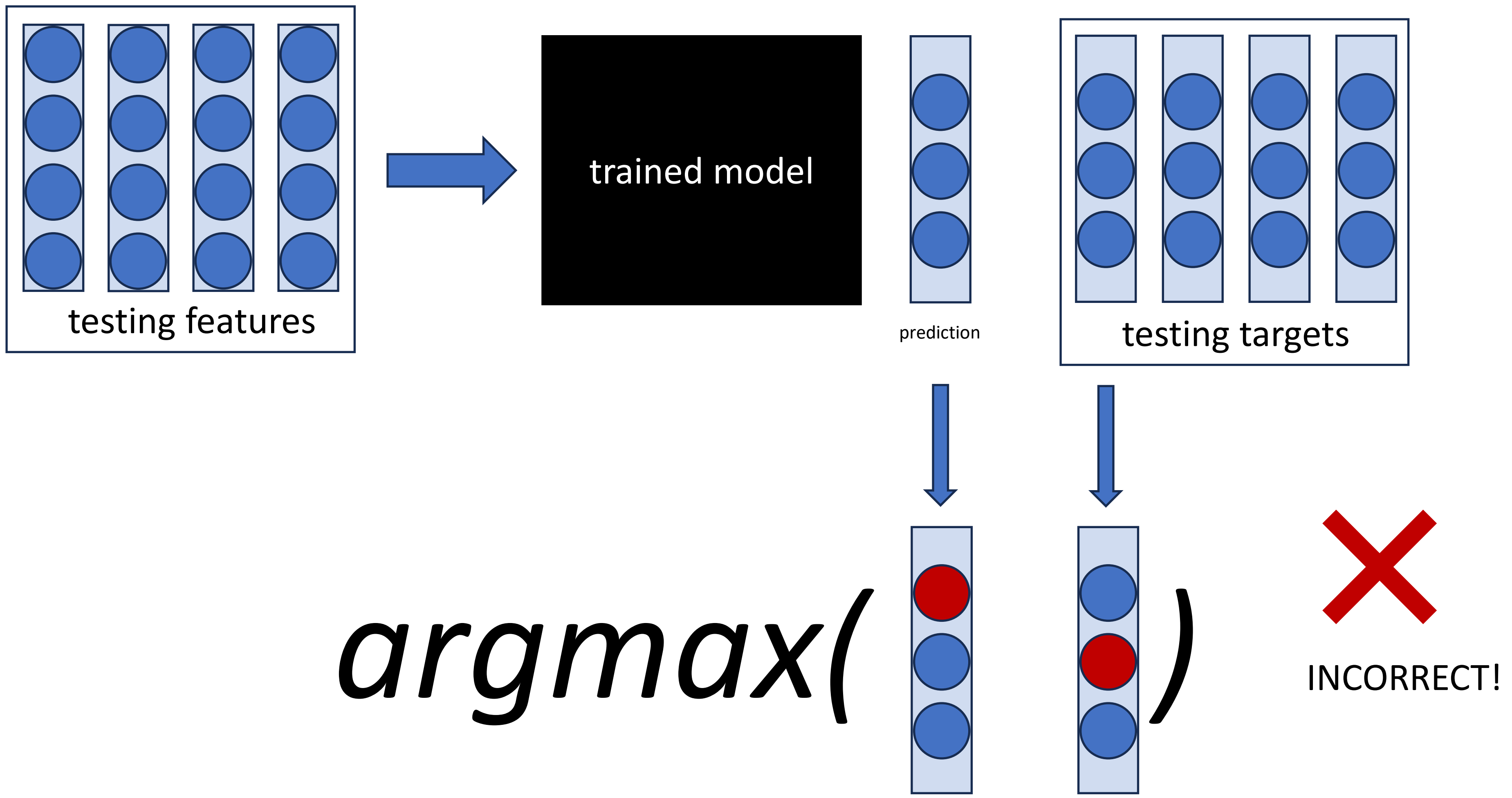

testing features
trained model

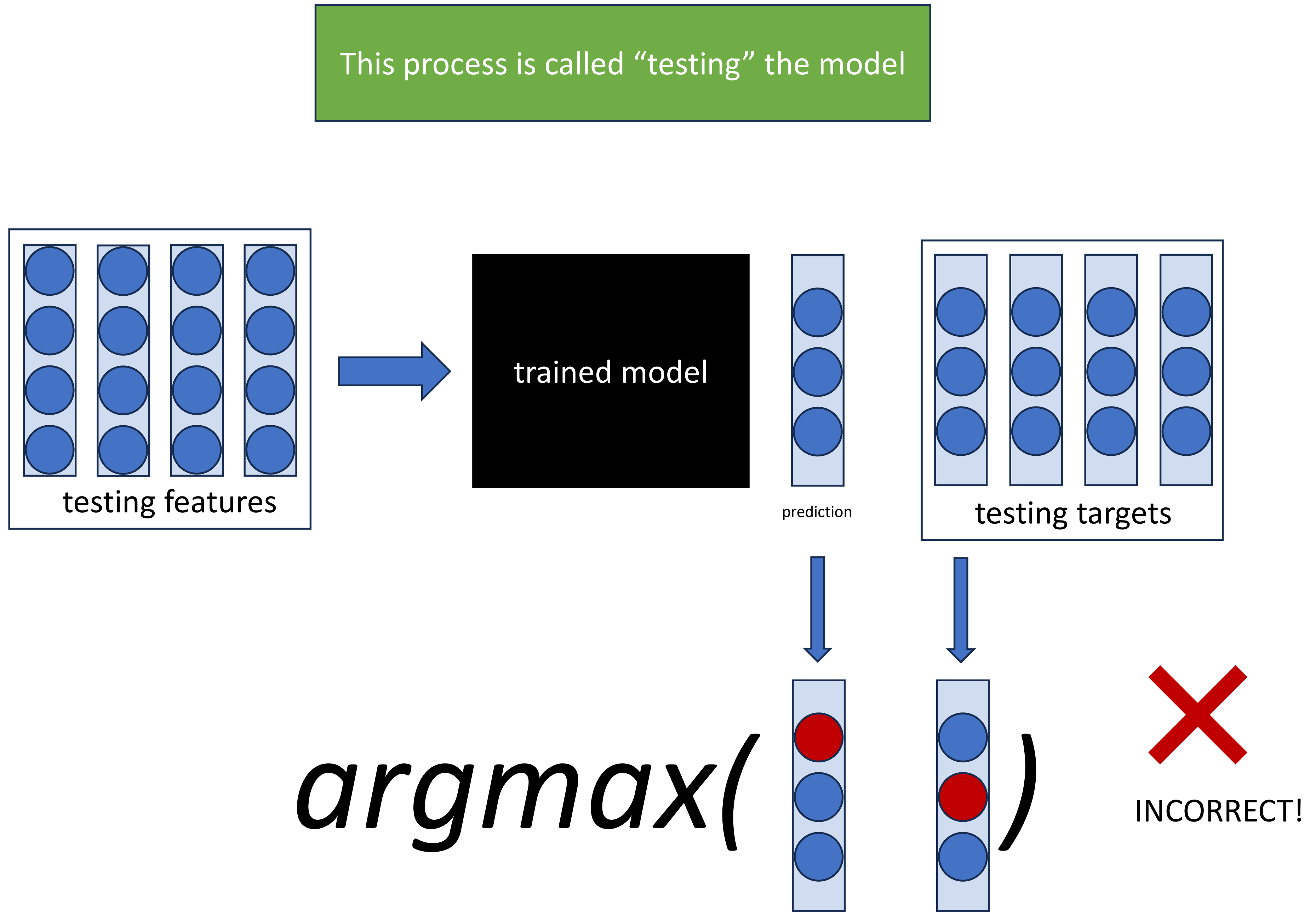

testing features
trained model
prediction


testing features
trained model
prediction
testing targets


testing features
trained model
prediction
testing targets
argmax( )


testing features
trained model
prediction
testing targets
argmax( )
CORRECT!


testing features
trained model
prediction
testing targets
argmax(
)
INCORRECT!


This process is called “testing” the model
testing features
trained model
prediction
testing targets
argmax( )
INCORRECT!

The training data represents a function. The job of the model is to approximate that function. We say that we are "fitting" the model to the data.

The training data represents a function. The job of the model is to approximate that function. We say that we are "fitting" the model to the data.

Sometimes, the model can become too "comfortable" with the training data. It will perform well when making predictions with the data it was trained on, but poorly on data it has never seen.

The training data represents a function. The job of the model is to approximate that function. We say that we are “fitting” the model to the data.

Sometimes, the model can become too “comfortable” with the training data. It will perform well when making predictions with the data it was trained on, but poorly on data it has never seen.

When this happens, it is called “overfitting”.

One way to prevent overfitting is by adding features to the training data. However, if these features are irrelevant, the model will not perform well. Select features is known as “feature engineering”.

One way to prevent overfitting is by adding features to the training data. However, if these features are irrelevant, the model will not perform well. Select features is known as "feature engineering".

Another technique for preventing overfitting is "dropout" when a portion of nodes are randomly "turned off".

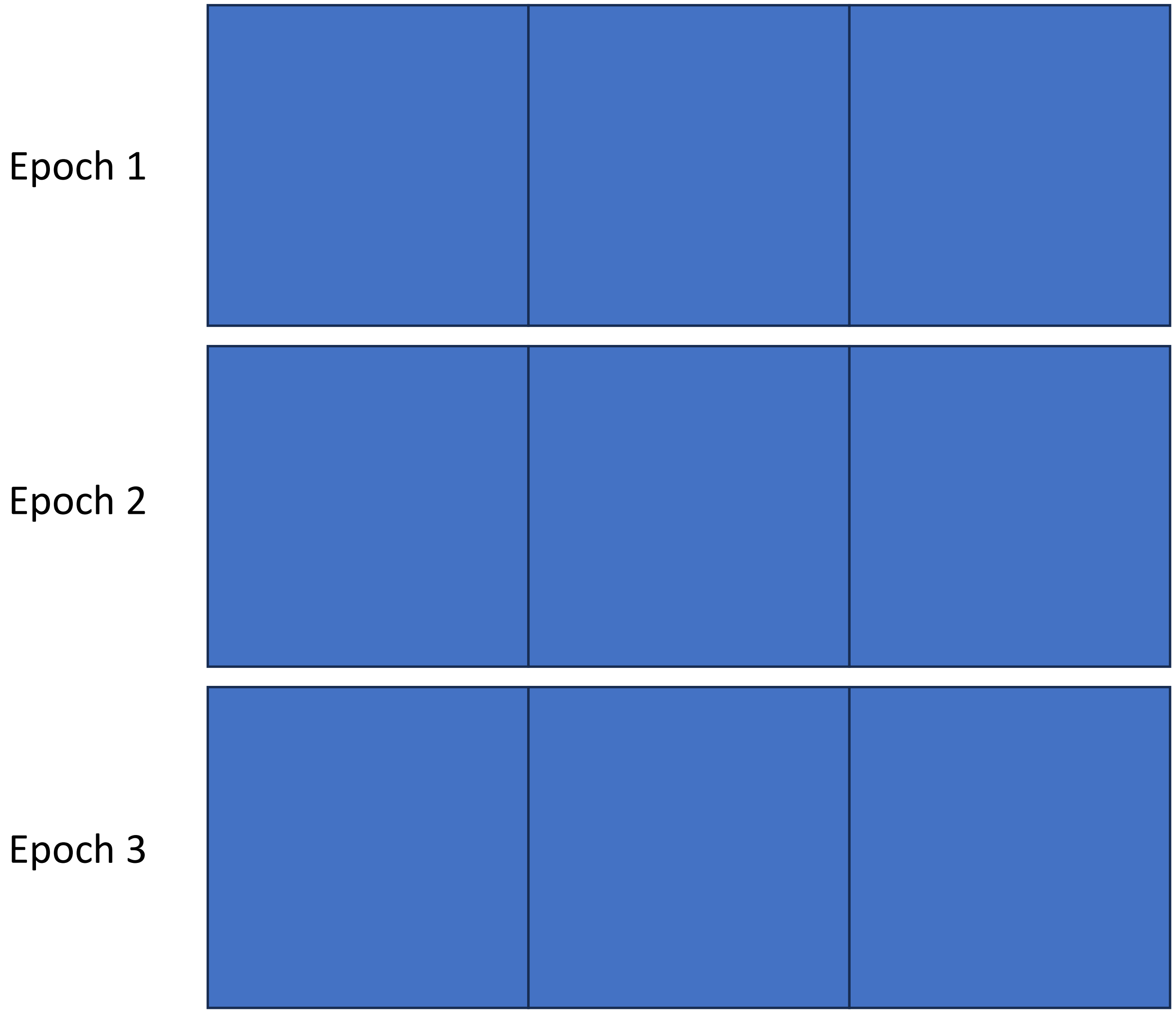
One way to prevent overfitting is by adding features to the training data. However, if these features are irrelevant, the model will not perform well. Select features is known as “feature engineering”.

Another technique for preventing overfitting is “dropout” when a portion of nodes are randomly “turned off”.

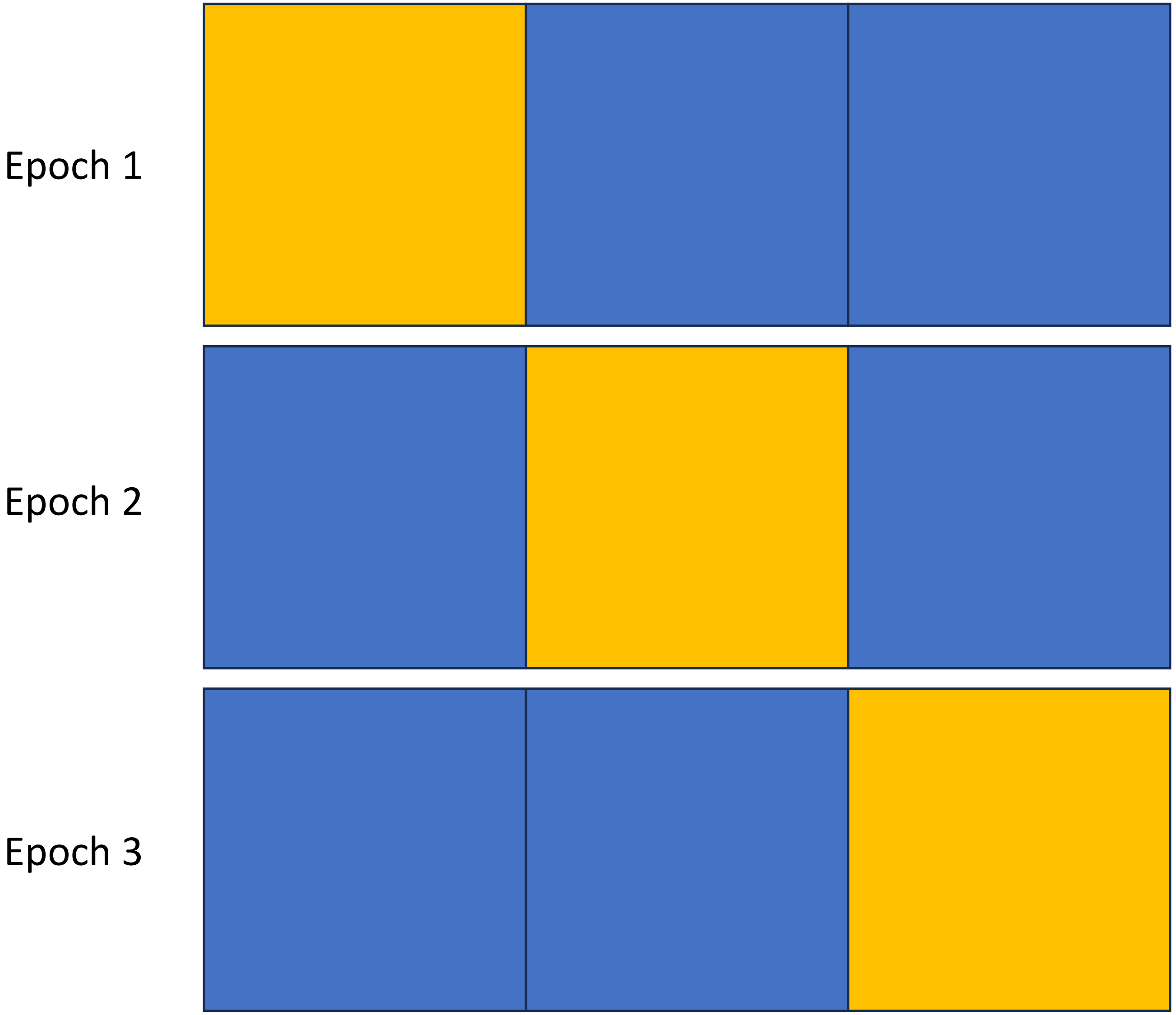
Finally, a similar concept called “k-fold” can be applied to the data.

training data

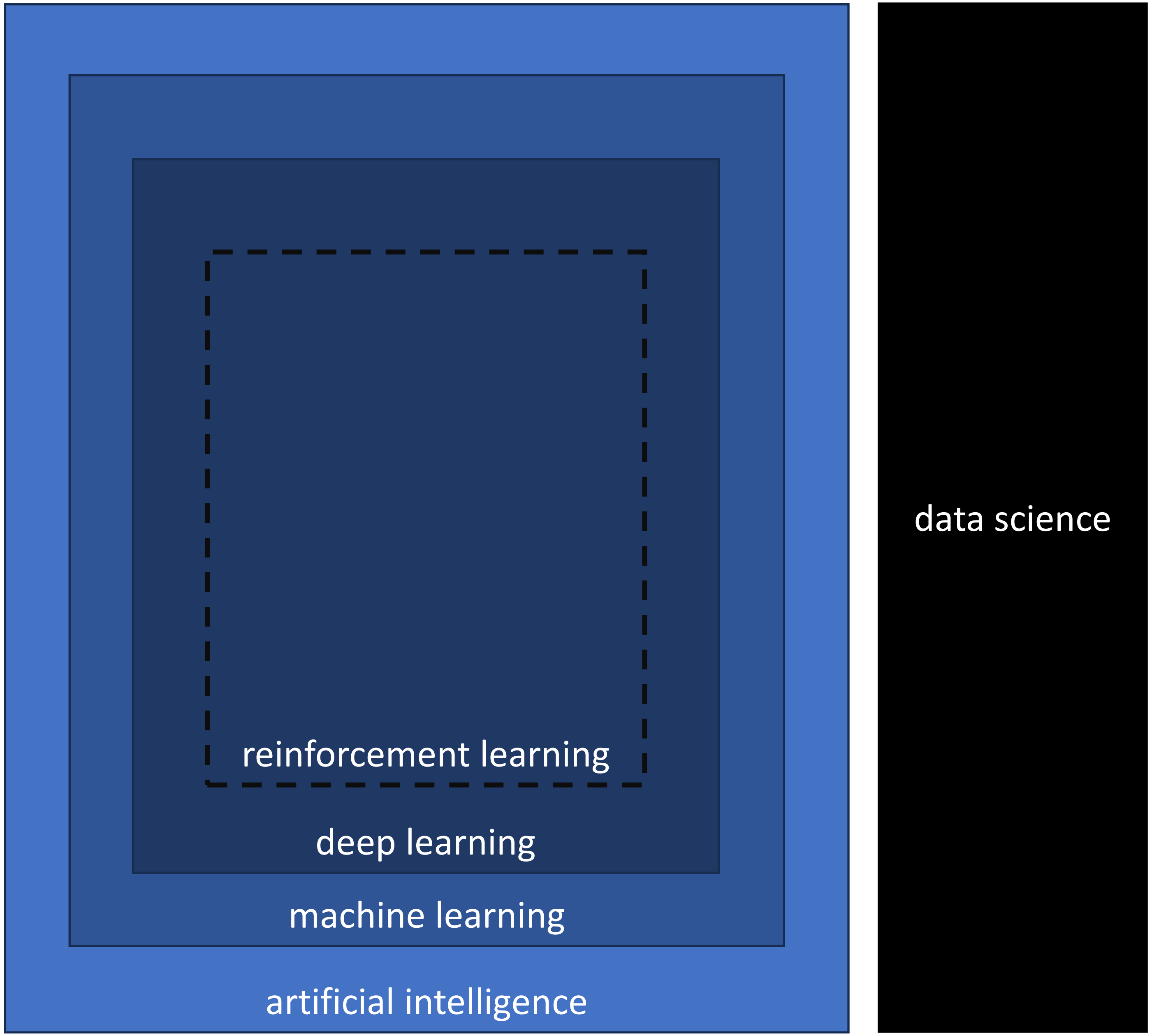

The training data will be analyzed by the network multiple times. Each time is called an epoch.
Epoch 1
training data
Epoch 2
training data
Epoch 3
training data



Here is how "3-fold" validation might work.



One fold is "turned off" during each epoch giving the network a more diverse view of the training data.


reinforcement learning
deep learning
machine learning
artificial intelligence
data science

TensorFlow

A Python package for implementing deep learning

TensorFlow

A Python package for implementing deep learning

But the core TensorFlow API is ugly

TensorFlow

A Python package for implementing deep learning

But the core TensorFlow API is ugly

And I mean ugly

TensorFlow

A Python package for implementing deep learning

But the core TensorFlow API is ugly

And I mean ugly

Really ugly

TensorFlow

A Python package for implementing deep learning

But the core TensorFlow API is ugly

And I mean ugly

Really ugly

Then a 3rd party project named Keras created a friendlier API that wraps the core API

TensorFlow

- A Python package for implementing deep learning
- But the core TensorFlow API is ugly
- And I mean ugly
- Really ugly
- Then a 3rd party project named Keras created a friendlier API that wraps the core API
- In TensorFlow 2, Keras became part of TensorFlow

TensorFlow

Grab the tensorflow module

TensorFlow

Grab the tensorflow module

```
import tensorflow as tf
```

TensorFlow

Grab the tensorflow module

And matplotlib so we can see the images

```
import tensorflow as tf
```

TensorFlow

Grab the tensorflow module

```
import tensorflow as tf
```

And matplotlib so we can see the images

```
import matplotlib.pyplot as plt
```

TensorFlow

Grab the tensorflow module

```
import tensorflow as tf
```

And matplotlib so we can see the images

```
import matplotlib.pyplot as plt
```

Fashion MNIST is included with Keras

TensorFlow

Grab the tensorflow module

```
import tensorflow as tf
```

And matplotlib so we can see the images

```
import matplotlib.pyplot as plt
```

Fashion MNIST is included with Keras

```
fashion_mnist = tf.keras.datasets.fashion_mnist
```

TensorFlow

Grab the tensorflow module

```
import tensorflow as tf
```

And matplotlib so we can see the images

```
import matplotlib.pyplot as plt
```

Fashion MNIST is included with Keras

```
fashion_mnist = tf.keras.datasets.fashion_mnist
```

Loading the data splits it into two 2-tuples

TensorFlow

Grab the tensorflow module

```
import tensorflow as tf
```

And matplotlib so we can see the images

```
import matplotlib.pyplot as plt
```

Fashion MNIST is included with Keras

```
fashion_mnist = tf.keras.datasets.fashion_mnist
```

Loading the data splits it into two 2-tuples

```
(X_train, y_train), (X_test, y_test) =
  fashion_mnist.load_data()
```

TensorFlow

Grab the tensorflow module

```
import tensorflow as tf
```

And matplotlib so we can see the images

```
import matplotlib.pyplot as plt
```

Fashion MNIST is included with Keras

```
fashion_mnist = tf.keras.datasets.fashion_mnist
```

Loading the data splits it into two 2-tuples

```
(X_train, y_train), (X_test, y_test) =
  fashion_mnist.load_data()
```

Take a look at the first image (features)

TensorFlow

Grab the tensorflow module

```
import tensorflow as tf
```

And matplotlib so we can see the images

```
import matplotlib.pyplot as plt
```

Fashion MNIST is included with Keras

```
fashion_mnist = tf.keras.datasets.fashion_mnist
```

Loading the data splits it into two 2-tuples

```
(X_train, y_train), (X_test, y_test) =
  fashion_mnist.load_data()
```

Take a look at the first image (features)

```
plt.imshow(X_train[0], cmap=“gray”)
```

TensorFlow

Grab the tensorflow module	`import tensorflow as tf`
And matplotlib so we can see the images	`import matplotlib.pyplot as plt`
Fashion MNIST is included with Keras	`fashion_mnist = tf.keras.datasets.fashion_mnist`
Loading the data splits it into two 2-tuples	`(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()`
Take a look at the first image (features)	`plt.imshow(X_train[0], cmap="gray")`
And the first target	

```
import tensorflow as tf

import matplotlib.pyplot as plt

fashion_mnist = tf.keras.datasets.fashion_mnist

(X_train, y_train), (X_test, y_test) =
  fashion_mnist.load_data()

plt.imshow(X_train[0], cmap=“gray”)
```

TensorFlow

Grab the tensorflow module	`import tensorflow as tf`
And matplotlib so we can see the images	`import matplotlib.pyplot as plt`
Fashion MNIST is included with Keras	`fashion_mnist = tf.keras.datasets.fashion_mnist`
Loading the data splits it into two 2-tuples	`(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()`
Take a look at the first image (features)	`plt.imshow(X_train[0], cmap="gray")`
And the first target	`y_train[0]`

TensorFlow

Grab the tensorflow module

```
import tensorflow as tf
```

And matplotlib so we can see the images

```
import matplotlib.pyplot as plt
```

Fashion MNIST is included with Keras

```
fashion_mnist = tf.keras.datasets.fashion_mnist
```

Loading the data splits it into two 2-tuples

```
(X_train, y_train), (X_test, y_test) =
  fashion_mnist.load_data()
```

Take a look at the first image (features)

```
plt.imshow(X_train[0], cmap=“gray”)
```

And the first target

```
y_train[0]
```

This makes more sense when you know the class names

TensorFlow

Grab the tensorflow module

```
import tensorflow as tf
```

And matplotlib so we can see the images

```
import matplotlib.pyplot as plt
```

Fashion MNIST is included with Keras

```
fashion_mnist = tf.keras.datasets.fashion_mnist
```

Loading the data splits it into two 2-tuples

```
(X_train, y_train), (X_test, y_test) =
  fashion_mnist.load_data()
```

Take a look at the first image (features)

```
plt.imshow(X_train[0], cmap=“gray”)
```

And the first target

```
y_train[0]
```

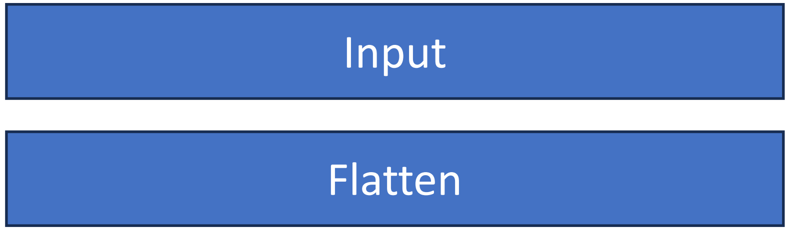
This makes more sense when you know the class names

```
classes = ["Top", "Trouser", "Pullover",
  "Dress", "Coat", "Sandal", "Shirt", "Sneaker",
  "Bag", "Ankle Boot"]
classes[y_train[0]]
```

TensorFlow

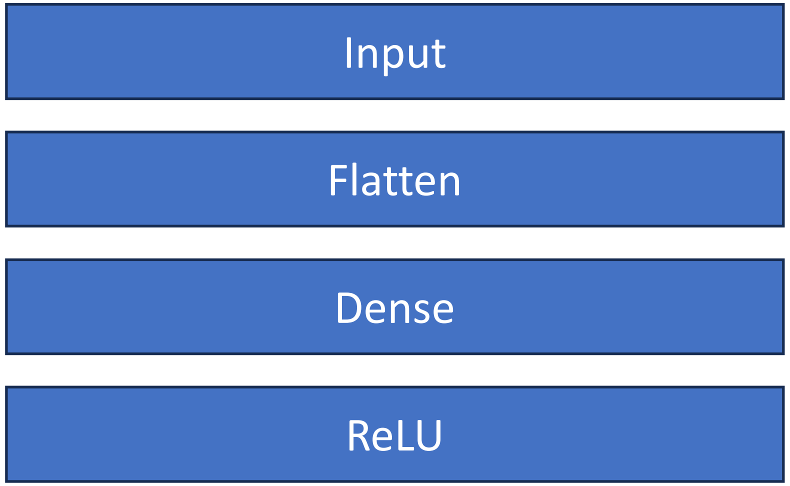
```
model = tf.keras.models.Sequential([

])
```

TensorFlow




```
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28,)),


])
```

TensorFlow




```
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28,)),
    tf.keras.layers.Dense(128, activation=tf.nn.relu),

])
```

Commercial Break: Activation layers

Every model tries to find a function to approximate the training data

Commercial Break: Activation layers

Every model tries to find a function to approximate the training data

Without activation layers, models can only approximate linear functions

Commercial Break: Activation layers

Every model tries to find a function to approximate the training data

Without activation layers, models can only approximate linear functions

Activation layers introduce non-linearity to make models universal function approximators

Commercial Break: Activation layers

Every model tries to find a function to approximate the training data

Without activation layers, models can only approximate linear functions

Activation layers introduce non-linearity to make models universal function approximators

There are many different activation functions including sigmoid, hyperbolic tangent and ReLU or Rectified Linear Unit.

Commercial Break: Activation layers

Every model tries to find a function to approximate the training data

Without activation layers, models can only approximate linear functions

Activation layers introduce non-linearity to make models universal function approximators

There are many different activation functions including sigmoid, hyperbolic tangent and ReLU or Rectified Linear Unit.

ReLU merely returns its input for non-negative numbers or 0.

Commercial Break: Activation layers

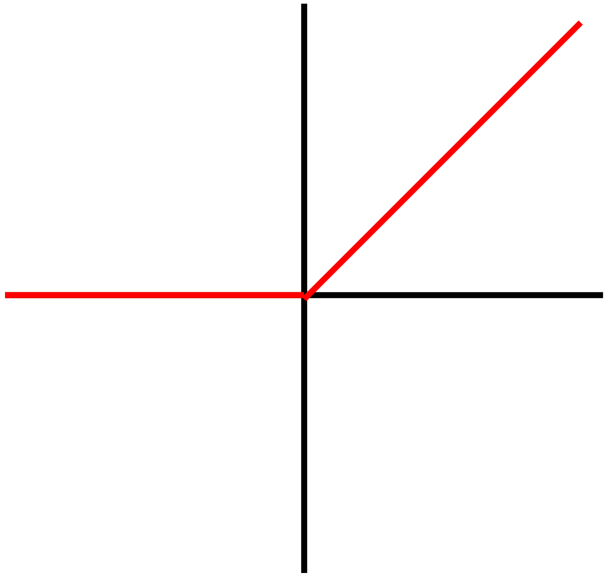
Every model tries to find a function to approximate the training data

Without activation layers, models can only approximate linear functions

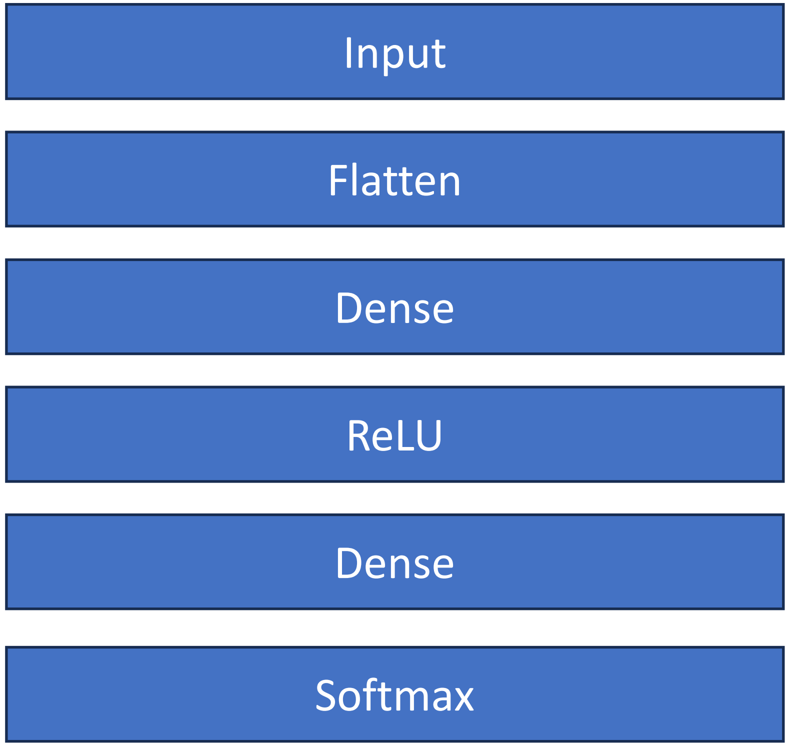
Activation layers introduce non-linearity to make models universal function approximators

There are many different activation functions including sigmoid, hyperbolic tangent and ReLU or Rectified Linear Unit.

ReLU merely returns its input for non-negative numbers or 0.



TensorFlow




```
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28,)),
    tf.keras.layers.Dense(128, activation=tf.nn.relu),
    tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
```

TensorFlow

See how many trainable parameters (weights) are in this toy dataset.

TensorFlow

See how many trainable parameters (weights) are in this toy dataset.

```
model.summary()
```

TensorFlow

See how many trainable parameters (weights) are in this toy dataset.

```
model.summary()
```

Call compile to add the loss function and optimizer

TensorFlow

See how many trainable parameters (weights) are in this toy dataset.

```
model.summary()
```

Call compile to add the loss function and optimizer

```
model.compile(
  loss="sparse_categorical_crossentropy",
  optimizer="adam", metrics=["accuracy"])
```

TensorFlow

See how many trainable parameters (weights) are in this toy dataset.

```
model.summary()
```

Call compile to add the loss function and optimizer

```
model.compile(
  loss="sparse_categorical_crossentropy",
  optimizer="adam", metrics=["accuracy"])
```

Fit the model to the training data over 5 epochs

TensorFlow

See how many trainable parameters (weights) are in this toy dataset.

```
model.summary()
```

Call compile to add the loss function and optimizer

```
model.compile(
  loss="sparse_categorical_crossentropy",
  optimizer="adam", metrics=["accuracy"])
```

Fit the model to the training data over 5 epochs

```
model.fit(X_train, y_train, epochs=5)
```

TensorFlow

See how many trainable parameters (weights) are in this toy dataset.

```
model.summary()
```

Call compile to add the loss function and optimizer

```
model.compile(
  loss="sparse_categorical_crossentropy",
  optimizer="adam", metrics=["accuracy"])
```

Fit the model to the training data over 5 epochs

```
model.fit(X_train, y_train, epochs=5)
```

This takes upwards of a minute. That’s not terribly long time to wait, but this is a toy dataset. Let’s speed things up using a GPU.

TensorFlow

See how many trainable parameters (weights) are in this toy dataset.

```
model.summary()
```

Call compile to add the loss function and optimizer

```
model.compile(
  loss="sparse_categorical_crossentropy",
  optimizer="adam", metrics=["accuracy"])
```

Fit the model to the training data over 5 epochs

```
model.fit(X_train, y_train, epochs=5)
```

This takes upwards of a minute. That's not terribly long time to wait, but this is a toy dataset. Let's speed things up using a GPU.

In CoLab, select from the menu Runtime -> Change runtime type

TensorFlow

See how many trainable parameters (weights) are in this toy dataset.

```
model.summary()
```

Call compile to add the loss function and optimizer

```
model.compile(
  loss="sparse_categorical_crossentropy",
  optimizer="adam", metrics=["accuracy"])
```

Fit the model to the training data over 5 epochs

```
model.fit(X_train, y_train, epochs=5)
```

This takes upwards of a minute. That's not terribly long time to wait, but this is a toy dataset. Let's speed things up using a GPU.

In CoLab, select from the menu Runtime -> Change runtime type

Click the radio button for T4 GPU and click the Save button

TensorFlow

See how many trainable parameters (weights) are in this toy dataset.

```
model.summary()
```

Call compile to add the loss function and optimizer

```
model.compile(
  loss="sparse_categorical_crossentropy",
  optimizer="adam", metrics=["accuracy"])
```

Fit the model to the training data over 5 epochs

```
model.fit(X_train, y_train, epochs=5)
```

This takes upwards of a minute. That's not terribly long time to wait, but this is a toy dataset. Let's speed things up using a GPU.

In CoLab, select from the menu Runtime -> Change runtime type

Click the radio button for T4 GPU and click the Save button

You'll need to rerun all the cells as the CPU runtime will be deleted. Select from the menu Runtime -> Run all

TensorFlow

Call evaluate with the testing data to see the test loss and accuracy

TensorFlow

Call evaluate with the testing data to see the test loss and accuracy

```
model.evaluate(X_test, y_test)
```

TensorFlow

Call evaluate with the testing data to see the test loss and accuracy

```
model.evaluate(X_test, y_test)
```

Get the predicted classes for the test images

TensorFlow

Call evaluate with the testing data to see the test loss and accuracy

```
model.evaluate(X_test, y_test)
```

Get the predicted classes for the test images

```
predictions = model.predict(X_test)
```

TensorFlow

Call evaluate with the testing data to see the test loss and accuracy

```
model.evaluate(X_test, y_test)
```

Get the predicted classes for the test images

```
predictions = model.predict(X_test)
```

Create a confusion matrix

TensorFlow

Call evaluate with the testing data to see the test loss and accuracy

```
model.evaluate(X_test, y_test)
```

Get the predicted classes for the test images

```
predictions = model.predict(X_test)
```

Create a confusion matrix

```
import numpy as np
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

TensorFlow

Call evaluate with the testing data to see the test loss and accuracy

```
model.evaluate(X_test, y_test)
```

Get the predicted classes for the test images

```
predictions = model.predict(X_test)
```

Create a confusion matrix

```
import numpy as np
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

The predictions have 10 values (a confidence score for each target) use the argmax function to find the largest

TensorFlow

Call evaluate with the testing data to see the test loss and accuracy

```
model.evaluate(X_test, y_test)
```

Get the predicted classes for the test images

```
predictions = model.predict(X_test)
```

Create a confusion matrix

```
import numpy as np
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

The predictions have 10 values (a confidence score for each target) use the argmax function to find the largest

```
cm = confusion_matrix(y_test, np.argmax(predictions, axis=1))
cmd = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=classes)
cmd.plot(cmap="gray")
```

TensorFlow

The convolutional neural network introduces two new layers

The convolutional layer applies a filter to detect features in the data

image

1	1	2	1	1	1
2	2	2	2	1	0
0	2	1	2	0	0
2	0	1	1	1	2
2	2	2	0	0	0
2	1	2	0	1	0

filter

0	0	0
1	1	1
0	0	0

image

1*0	1*0	2*0	1	1	1
2*1	2*1	2*1	2	1	0
0*0	2*0	1*0	2	0	0
2	0	1	1	1	2
2	2	2	0	0	0
2	1	2	0	1	0

filter

0	0	0
1	1	1
0	0	0

convolution

6

image

1	1	2	1	1	1
2	2	2	2	1	0
0	2	1	2	0	0
2	0	1	1	1	2
2	2	2	0	0	0
2	1	2	0	1	0

filter

0	0	0
1	1	1
0	0	0

convolution

6	6

image

1	1	2	1	1	1
2	2	2	2	1	0
0	2	1	2	0	0
2	0	1	1	1	2
2	2	2	0	0	0
2	1	2	0	1	0

filter

0	0	0
1	1	1
0	0	0

convolution

6	6	5

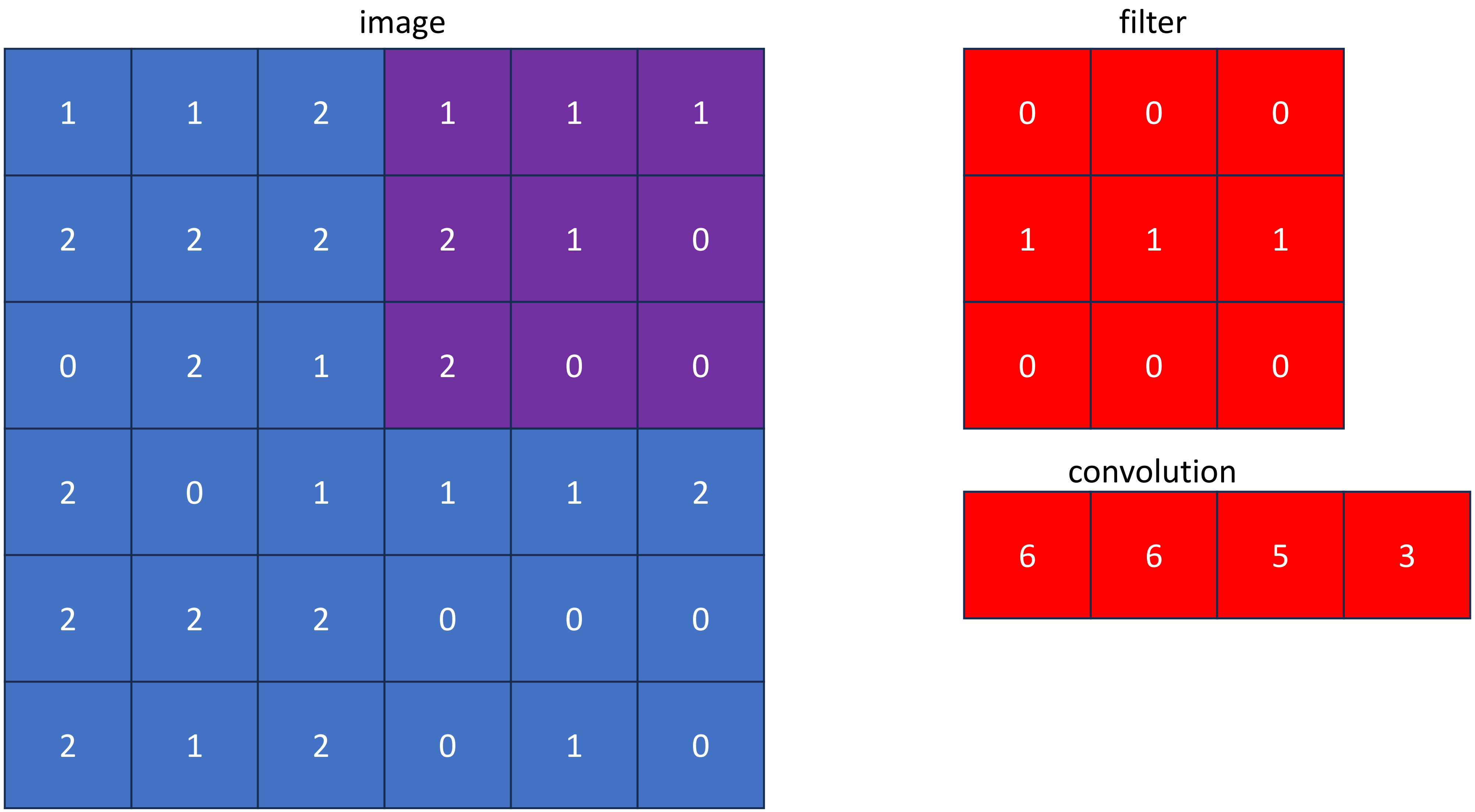


image
1 1 2 1 1 1
2 2 2 2 1 0
0 2 1 2 0 0
2 0 1 1 1 2
2 2 2 0 0 0
2 1 2 0 1 0
filter
0 0 0
1 1 1
0 0 0
convolution
6 6 5 3

image

1	1	2	1	1	1
2	2	2	2	1	0
0	2	1	2	0	0
2	0	1	1	1	2
2	2	2	0	0	0
2	1	2	0	1	0

filter

0	0	0
1	1	1
0	0	0

convolution

6	6	5	3
3			

image

1	1	2	1	1	1
2	2	2	2	1	0
0	2	1	2	0	0
2	0	1	1	1	2
2	2	2	0	0	0
2	1	2	0	1	0

filter

0	0	0
1	1	1
0	0	0

convolution

6	6	5	3
3	5	3	4
3	2	3	4
6	4	2	0

TensorFlow

The convolutional neural network introduces two new layers

The convolutional layer applies a filter to detect features in the data

The pooling layer downsamples the data

TensorFlow

The convolutional neural network introduces two new layers

The convolutional layer applies a filter to detect features in the data

The pooling layer downsamples the data

You can use max or average pooling

convolution

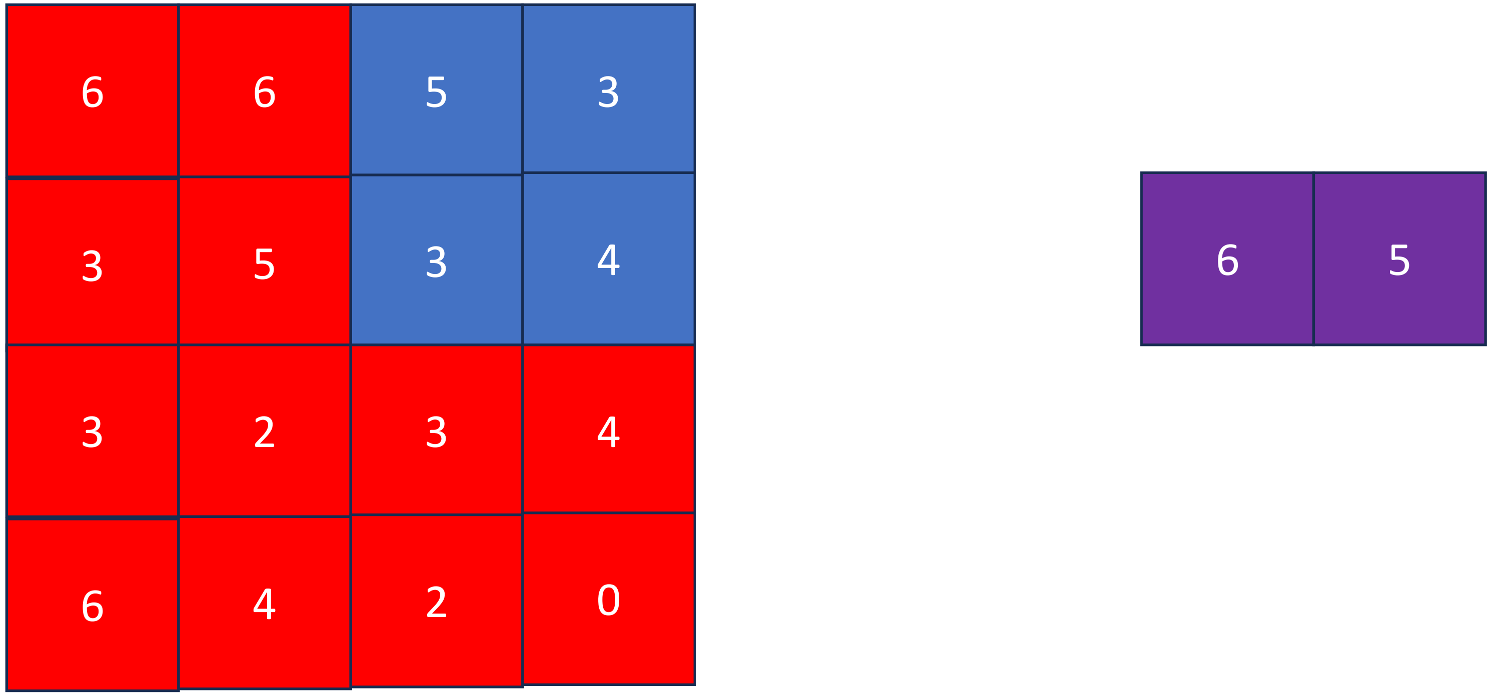
6	6	5	3
3	5	3	4
3	2	3	4
6	4	2	0

convolution

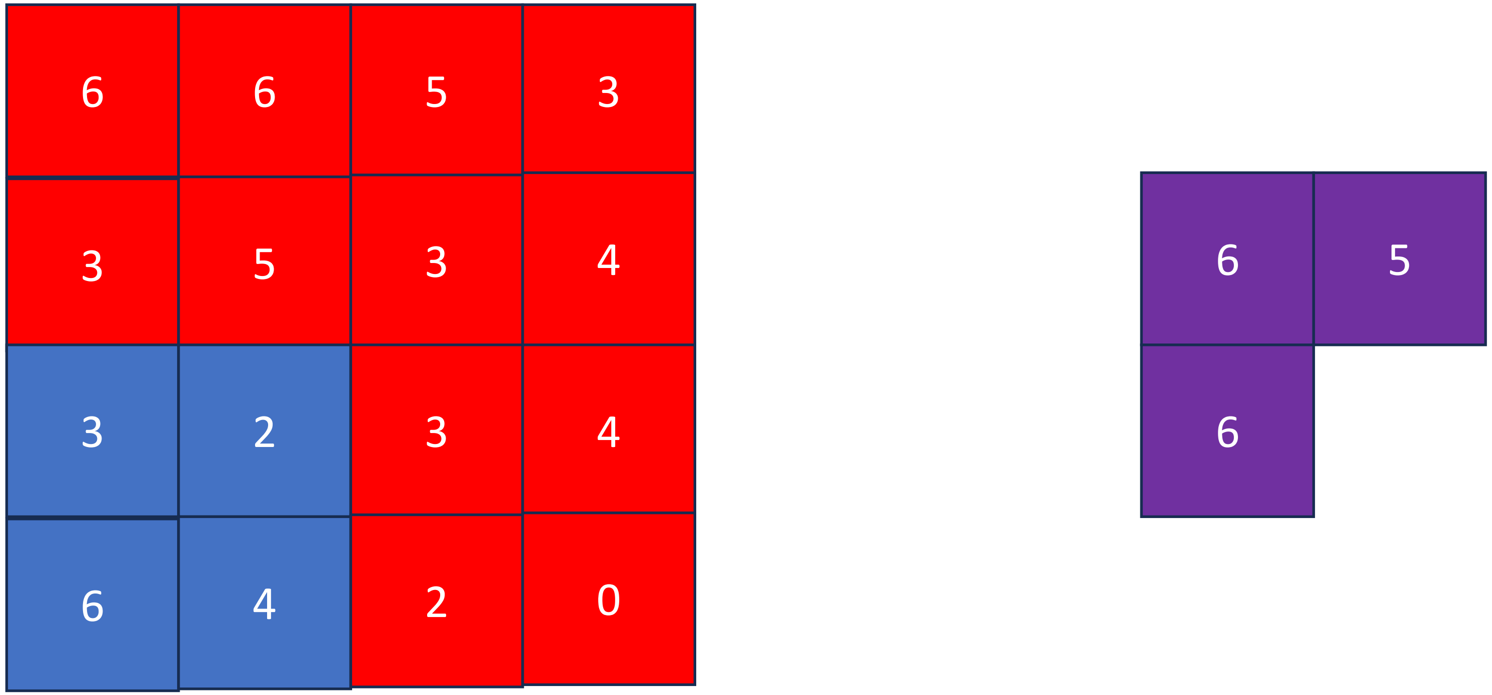
6	6	5	3
3	5	3	4
3	2	3	4
6	4	2	0

6

convolution



convolution



convolution

6	6	5	3
3	5	3	4
3	2	3	4
6	4	2	0

6	5
6	4

TensorFlow

The convolutional neural network introduces two new layers

The convolutional layer applies a filter to detect features in the data

The pooling layer downsamples the data

You can use max or average pooling

The output of the pooling layer is flattened before going to one or more dense layers

```
model = tf.keras.models.Sequential([
    layers.Convolution2D(32, (3, 3,), activation="relu", input_shape=(32, 32, 3,)),
    layers.MaxPooling2D(pool_size=(2, 2,)),
    layers.Dropout(0.25),
    layers.Flatten(),
    layers.Dense(512, activation="relu"),
    layers.Dropout(0.5),
    layers.Dense(10, activation="softmax"),
])
```

The convolution layer will have 32 3x3 filters

```
model = tf.keras.models.Sequential([
    layers.Convolution2D(32, (3, 3,), activation="relu", input_shape=(32, 32, 3,)),
    layers.MaxPooling2D(pool_size=(2, 2,)),
    layers.Dropout(0.25),
    layers.Flatten(),
    layers.Dense(512, activation="relu"),
    layers.Dropout(0.5),
    layers.Dense(10, activation="softmax"),
])
```

The pooling layer will use a 2x2 window and select the maximum value

```
model = tf.keras.models.Sequential([
    layers.Convolution2D(32, (3, 3,), activation="relu", input_shape=(32, 32, 3,)),
    layers.MaxPooling2D(pool_size=(2, 2,)),
    layers.Dropout(0.25),
    layers.Flatten(),
    layers.Dense(512, activation="relu"),
    layers.Dropout(0.5),
    layers.Dense(10, activation="softmax"),
])
```

And this is the first time we have seen Dropout layers

```
model = tf.keras.models.Sequential([
    layers.Convolution2D(32, (3, 3,), activation="relu", input_shape=(32, 32, 3,)),
    layers.MaxPooling2D(pool_size=(2, 2,)),
    layers.Dropout(0.25),
    layers.Flatten(),
    layers.Dense(512, activation="relu"),
    layers.Dropout(0.5),
    layers.Dense(10, activation="softmax"),
])
```

TensorBoard

Visualization tool for inspecting models

TensorBoard

Visualization tool for inspecting models

Draw graphs of loss and accuracy and other values

TensorBoard

- Visualization tool for inspecting models
- Draw graphs of loss and accuracy and other values
- View training data of images and audio

TensorBoard

- Visualization tool for inspecting models
- Draw graphs of loss and accuracy and other values
- View training data of images and audio
- Profiler

TensorBoard

- Visualization tool for inspecting models
- Draw graphs of loss and accuracy and other values
- View training data of images and audio
- Profiler
- Debugger

TensorBoard

- Visualization tool for inspecting models
- Draw graphs of loss and accuracy and other values
- View training data of images and audio
- Profiler
- Debugger
- Runs standalone on the desktop or in Jupyter Notebook

TensorBoard

- Visualization tool for inspecting models
- Draw graphs of loss and accuracy and other values
- View training data of images and audio
- Profiler
- Debugger
- Runs standalone on the desktop or in Jupyter Notebook
- Works with machine learning frameworks other than TensorFlow and Keras (PyTorch)

TensorBoard

Load the Jupyter Notebook extension

TensorBoard

Load the Jupyter Notebook extension

```
%load_ext tensorboard
```

TensorBoard

Load the Jupyter Notebook extension

```
%load_ext tensorboard
```

Create a callback

TensorBoard

Load the Jupyter Notebook extension

```
%load_ext tensorboard
```

Create a callback

```
tf.keras.callbacks.TensorBoard(log_dir=“logs”)
```

TensorBoard

Load the Jupyter Notebook extension	`%load_ext tensorboard`
Create a callback	`tf.keras.callbacks.TensorBoard(log_dir=“logs”)`
Call the fit method and pass it the callback	

TensorBoard

Load the Jupyter Notebook extension	`%load_ext tensorboard`
Create a callback	`tf.keras.callbacks.TensorBoard(log_dir=“logs”)`
Call the fit method and pass it the callback	`model.fit(..., callbacks=[callback])`

TensorBoard

Load the Jupyter Notebook extension

```
%load_ext tensorboard
```

Create a callback

```
tf.keras.callbacks.TensorBoard(log_dir=“logs”)
```

Call the fit method and pass it the callback

```
model.fit(..., callbacks=[callback])
```

Start TensorBoard (in Jupyter Notebook)

TensorBoard

Load the Jupyter Notebook extension	`%load_ext tensorboard`
Create a callback	`tf.keras.callbacks.TensorBoard(log_dir=“logs”)`
Call the fit method and pass it the callback	`model.fit(..., callbacks=[callback])`
Start TensorBoard (in Jupyter Notebook)	`%tensorboard --logdir logs`

TensorBoard

Load the Jupyter Notebook extension	`%load_ext tensorboard`
Create a callback	`tf.keras.callbacks.TensorBoard(log_dir="logs")`
Call the fit method and pass it the callback	`model.fit(..., callbacks=[callback])`
Start TensorBoard (in Jupyter Notebook)	`%tensorboard --logdir logs`
Visualize the testing images	

TensorBoard

Load the Jupyter Notebook extension	`%load_ext tensorboard`
Create a callback	`tf.keras.callbacks.TensorBoard(log_dir=“logs”)`
Call the fit method and pass it the callback	`model.fit(..., callbacks=[callback])`
Start TensorBoard (in Jupyter Notebook)	`%tensorboard --logdir logs`
Visualize the testing images	
Create a file writer to write the test image data	

TensorBoard

Load the Jupyter Notebook extension	`%load_ext tensorboard`
Create a callback	`tf.keras.callbacks.TensorBoard(log_dir=“logs”)`
Call the fit method and pass it the callback	`model.fit(..., callbacks=[callback])`
Start TensorBoard (in Jupyter Notebook)	`%tensorboard --logdir logs`
Visualize the testing images	
Create a file writer to write the test image data	`fw = tf.summary.create_file_writer( "logs/images")`

TensorBoard

Load the Jupyter Notebook extension	`%load_ext tensorboard`
Create a callback	`tf.keras.callbacks.TensorBoard(log_dir=“logs”)`
Call the fit method and pass it the callback	`model.fit(..., callbacks=[callback])`
Start TensorBoard (in Jupyter Notebook)	`%tensorboard --logdir logs`
Visualize the testing images	
Create a file writer to write the test image data	`fw = tf.summary.create_file_writer( "logs/images")`
Reshape the image data to add a dimension (1 for monochrome)	

TensorBoard

Load the Jupyter Notebook extension	`%load_ext tensorboard`
Create a callback	`tf.keras.callbacks.TensorBoard(log_dir=“logs”)`
Call the fit method and pass it the callback	`model.fit(..., callbacks=[callback])`
Start TensorBoard (in Jupyter Notebook)	`%tensorboard --logdir logs`
Visualize the testing images	
Create a file writer to write the test image data	`fw = tf.summary.create_file_writer("logs/images")`
Reshape the image data to add a dimension (1 for monochrome)	`with fw.as_default():` `  images = X_test[:25].reshape(-1, 28, 28, 1)`

TensorBoard

Load the Jupyter Notebook extension

```
%load_ext tensorboard
```

Create a callback

```
tf.keras.callbacks.TensorBoard(log_dir=“logs”)
```

Call the fit method and pass it the callback

```
model.fit(..., callbacks=[callback])
```

Start TensorBoard (in Jupyter Notebook)

```
%tensorboard --logdir logs
```

Visualize the testing images

Create a file writer to write the test image data

```
fw = tf.summary.create_file_writer(
  "logs/images")
```

Reshape the image data to add a dimension (1 for monochrome)

```
with fw.as_default():
  images = X_test[:25].reshape(-1, 28, 28, 1)
```

Write the images

TensorBoard

Load the Jupyter Notebook extension	`%load_ext tensorboard`
Create a callback	`tf.keras.callbacks.TensorBoard(log_dir=“logs”)`
Call the fit method and pass it the callback	`model.fit(..., callbacks=[callback])`
Start TensorBoard (in Jupyter Notebook)	`%tensorboard --logdir logs`
Visualize the testing images	
Create a file writer to write the test image data	`fw = tf.summary.create_file_writer( "logs/images")`
Reshape the image data to add a dimension (1 for monochrome)	`with fw.as_default(): images = X_test[:25].reshape(-1, 28, 28, 1)`
Write the images	`tf.summary.image(“Training images, images max_outputs=25, step=0)`

TensorFlow Playground

playground.tensorflow.org

Any Questions?

Closing thoughts

- If you walk away with nothing else, outside of the “Data Science Profitability Path” remember these points
  - Jupyter Notebook
  - pandas
  - Matplotlib
- Tomorrow 8:15 AI for Everyone with Azure AI Services
- Please visit and thank the sponsors
- I will be around for the rest of the event if you’d like to talk
- Continue the conversation: linktr.ee/douglasstarnes (LinkedIn is best)

THANK YOU!