DeepLearning.AI

Why PyTorch?

Getting Started with PyTorch

Adding Two Numbers with PyTorch

```
result = a + b
```

```
a = torch.tensor(2.0)
b = torch.tensor(3.0)
result = a + b
print(result)
```

```
C:>
tensor(5.)
```

Machine Learning ≠ **Traditional Programming**

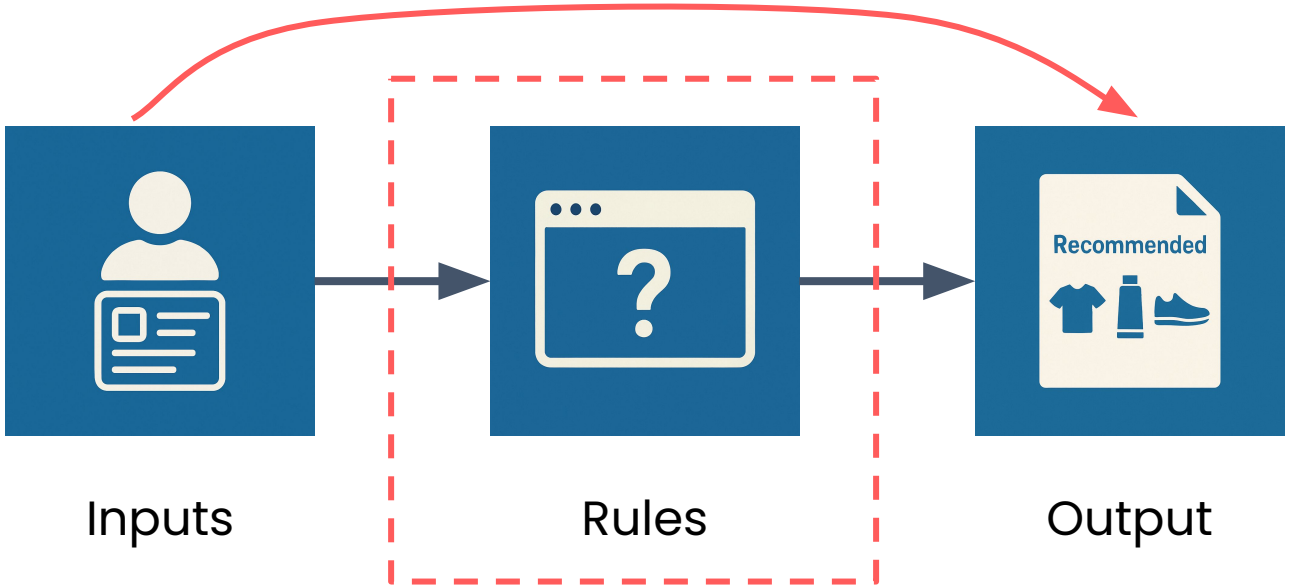
Product Recommendation

Product Recommendation

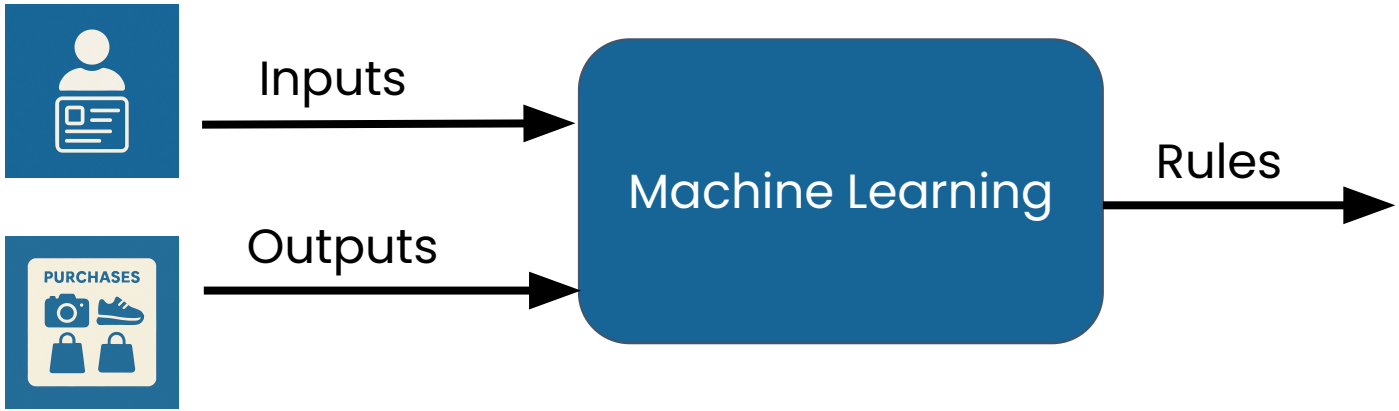
Rule: ***"If a customer buys a camera, recommend camera lenses"***

- What if it was a gift?
- What if they already own five lenses?

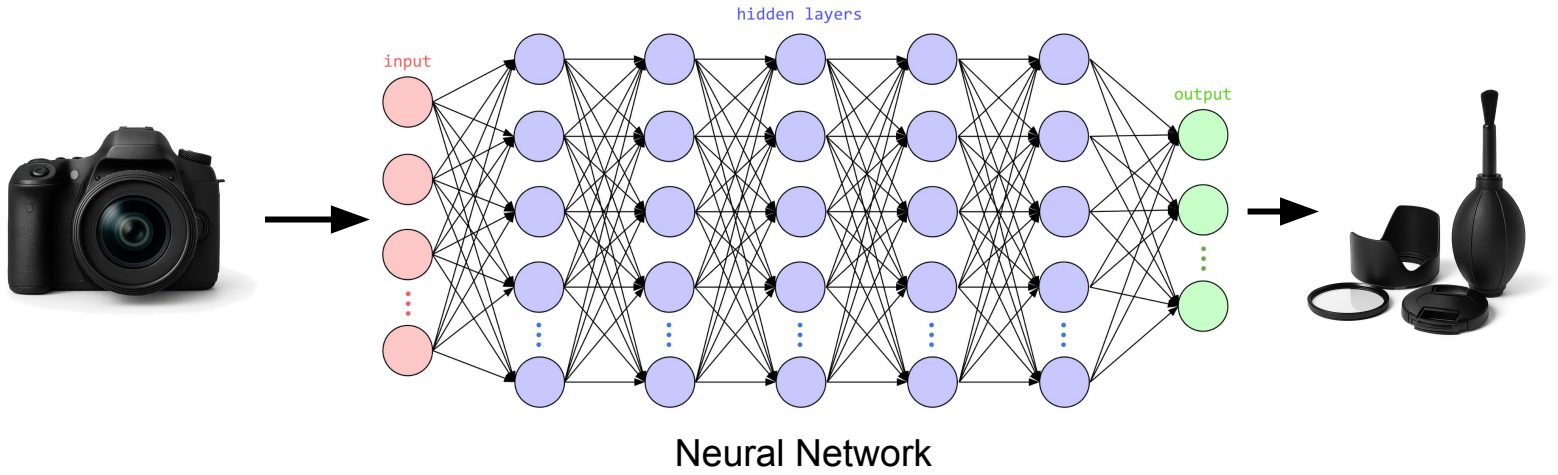
Traditional Programing



Machine Learning

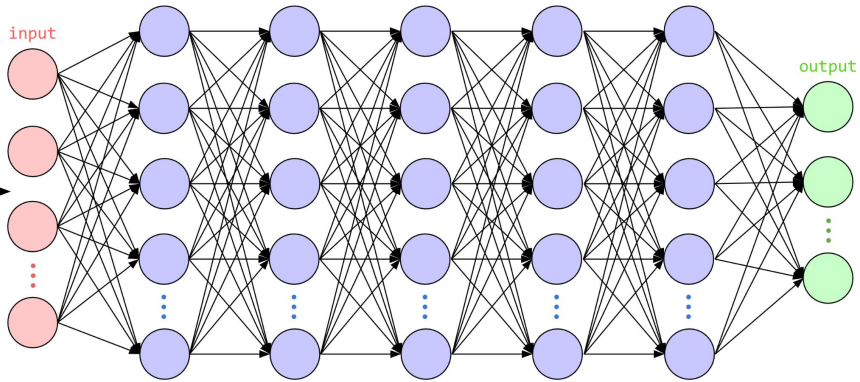


Deep Learning - Neural Networks



Deep Learning - Neural Networks

Cat Images



Neural Network

Cat

massive amount of calculations

Early Frameworks

```
result = a + b
```

```
a = framework.placeholder(float32)
b = framework.placeholder(float32)
result = framework.add(a,b)
with framework.Executor() as executor:
   actual_result = executor.run(result,
data_dictionary={{a: 2.0, b: 3.0}})
```

```
C:>
<Object 'Add:0' shape=(2,) dtype=float32>
```

Early Frameworks

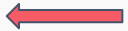
Static Computation Graph

Early Frameworks

```
# A chain of operations, one feeding into the next
result1 = do_math(input)
result2 = do_math(result1)
result3 = do_math(result2)
result4 = do_math(result3)
result5 = do_math(result4)
# ... and so on, down the line
```

Early Frameworks

```
# A chain of operations, one feeding into the next
result1 = do_math(input)
result2 = do_math(result1)

new_op = do_math(result2)

result3 = do_math(result2)
result4 = do_math(result3)
result5 = do_math(result4)
```



Early Frameworks

```
# A chain of operations, one feeding into the next
result1 = do_math(input)
result2 = do_math(result1)

new_op = do_math(result2)

result3 = do_math(result2)
result4 = do_math(result3)
result5 = do_math(result4)
```

No Testing of Parts

Early Frameworks

```
# A chain of operations, one feeding into the next
result1 = do_math(input)
result2 = do_math(result1)

new_op = do_math(result2)

result3 = do_math(result2)
result4 = do_math(result3)
result5 = do_math(result4)
```

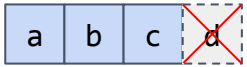
Full Build Required

Early Frameworks

~~`if a:`~~

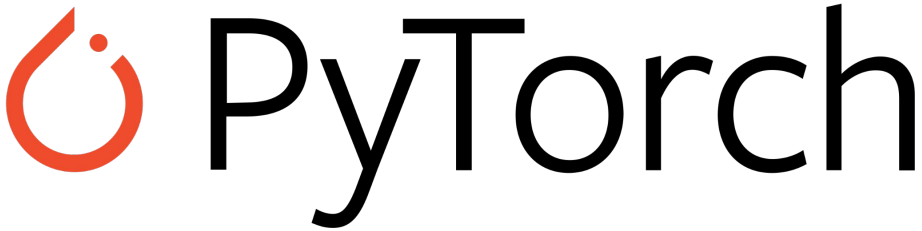
~~`for a in b:`~~

`Fixed input size`



~~debugging~~

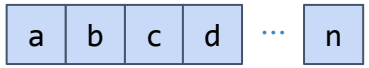
`untraceable system errors`


PyTorch

Early Frameworks

✔ `If a:`

✔ `for a in b:`

✔ 

✔ debugging

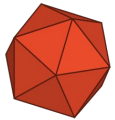
✔ In code errors

✔ Python-like code

PyTorch Ecosystem

SKORCH

OpenMined



PyTorch geometric

PyTorch

flair

PENNYLANE

BoTorch

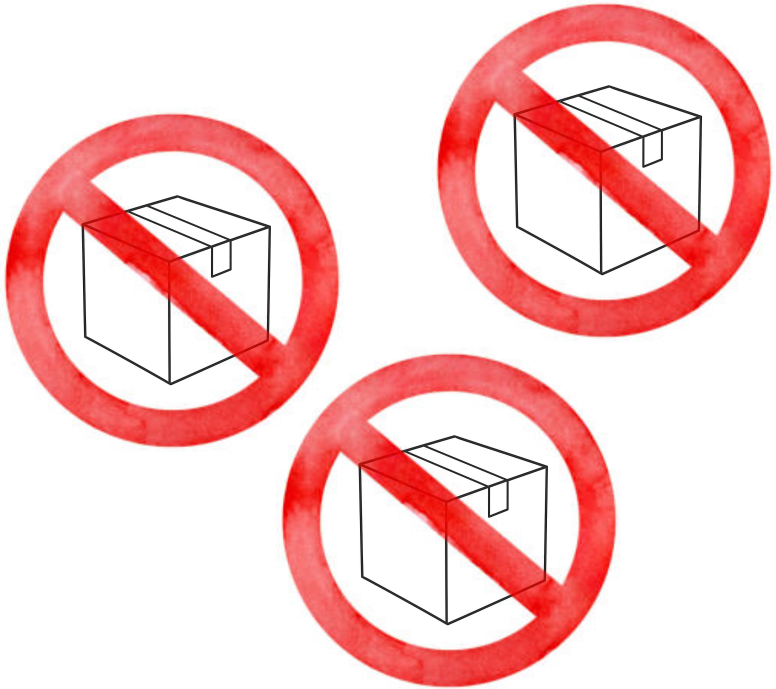
DeepLearning.AI

The Building Block of Neural Networks

Getting Started with PyTorch

The Delivery Problem

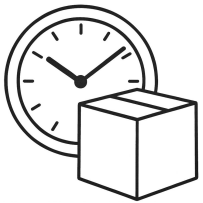
30 minute guarantee



The Delivery Problem

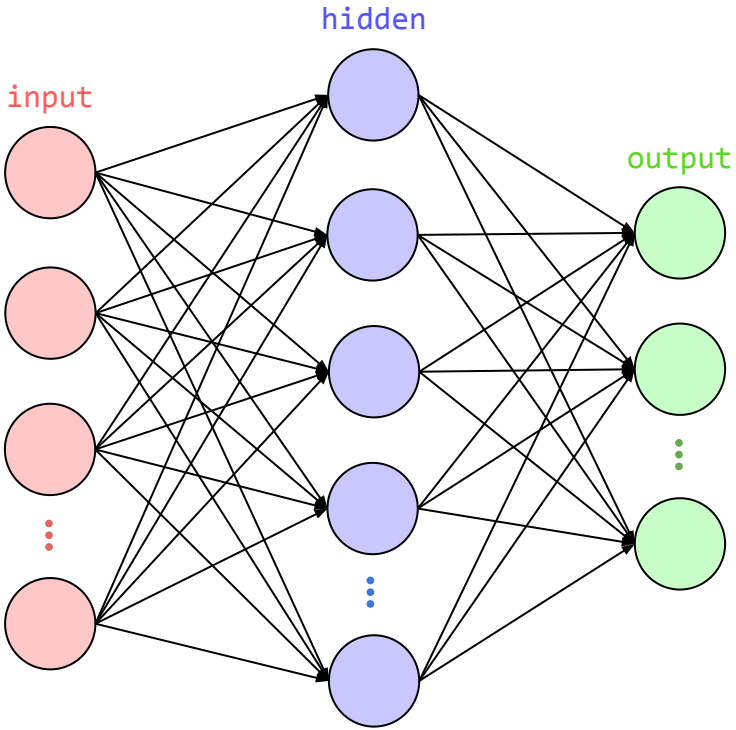
30 minute guarantee

Take the job?

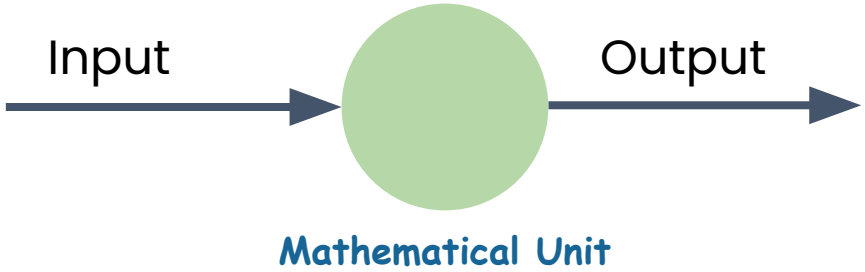


7 mile delivery

Neural Networks



Single Neuron



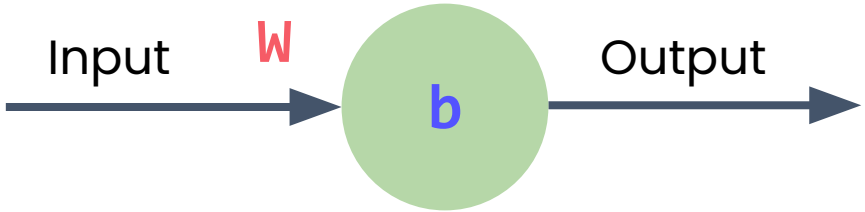
Delivery Data

Distance (miles)	Delivery Time (minutes)
2.0	11.9
3.0	15.3
4.0	18.8
5.0	22.2
6.0	25.6

Prediction:
7 miles

$$time = W * distance + b$$

Single Neuron



$$output = W * input + b$$

Single Neuron

$$time = 1.0 * distance + 10.0$$

Distance (miles)	Delivery Time (minutes)
2.0	11.9
3.0	15.3
4.0	18.8
5.0	22.2
6.0	25.6

Prediction:

`1.0*5 + 10.0 = 15`

Single Neuron

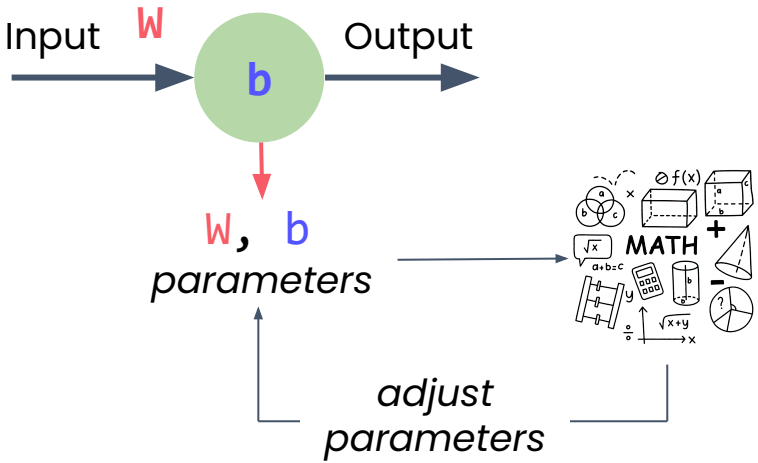
Distance (miles)	Delivery Time (minutes)
2.0	11.9
3.0	15.3
4.0	18.8
5.0	22.2
6.0	25.6

$$time = 3.4 * distance + 5.0$$

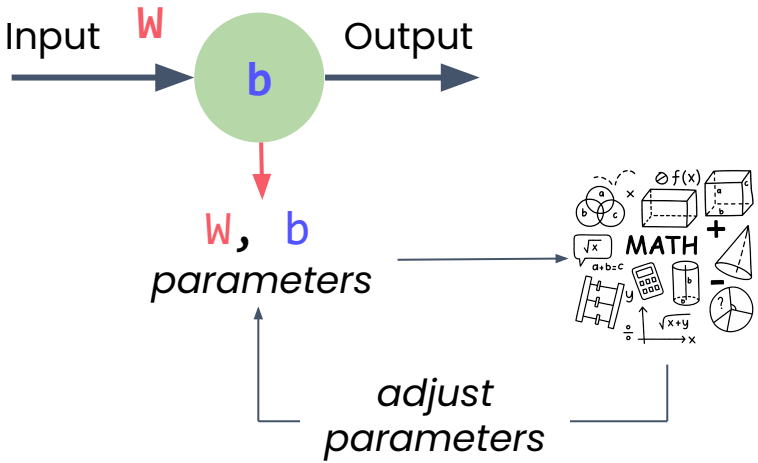
Prediction:

3.4*5 + 5.0 = 22

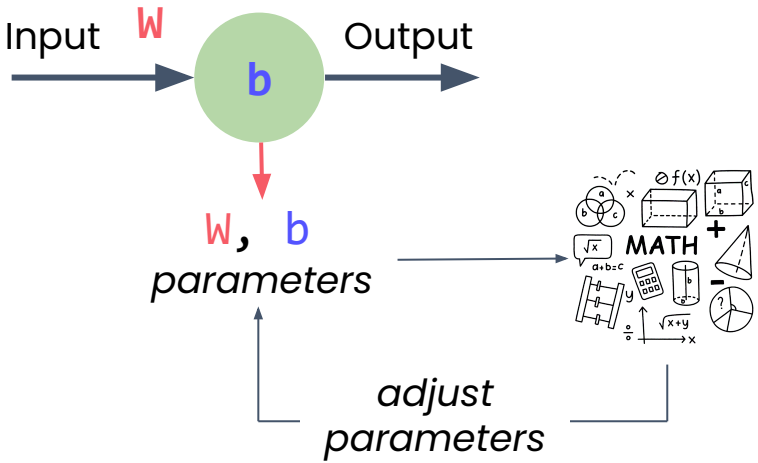
Single Neuron



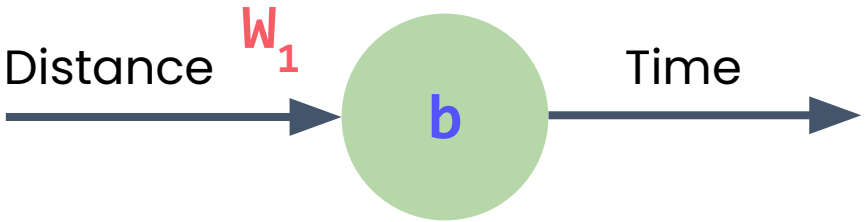
Single Neuron



Single Neuron

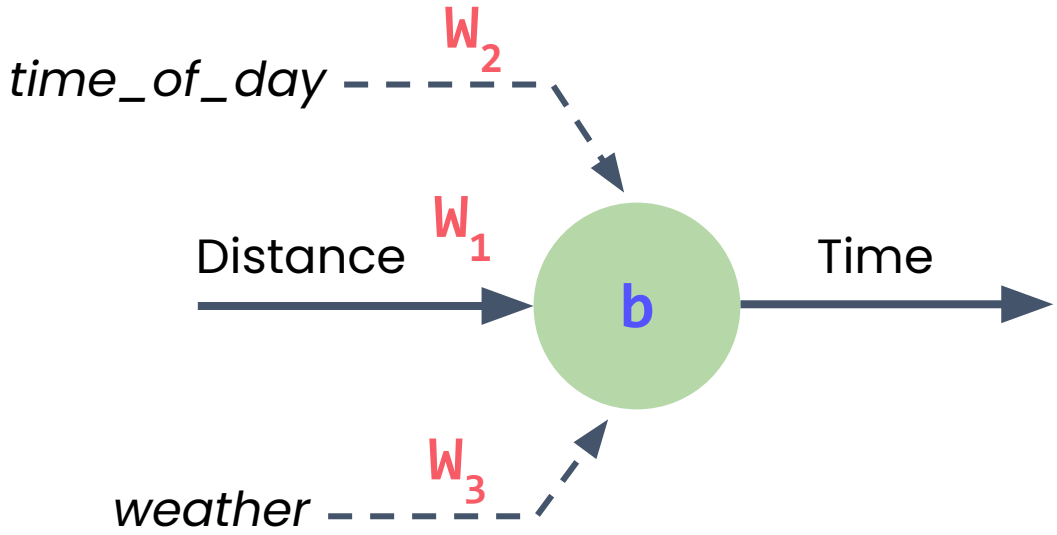


Single Neuron



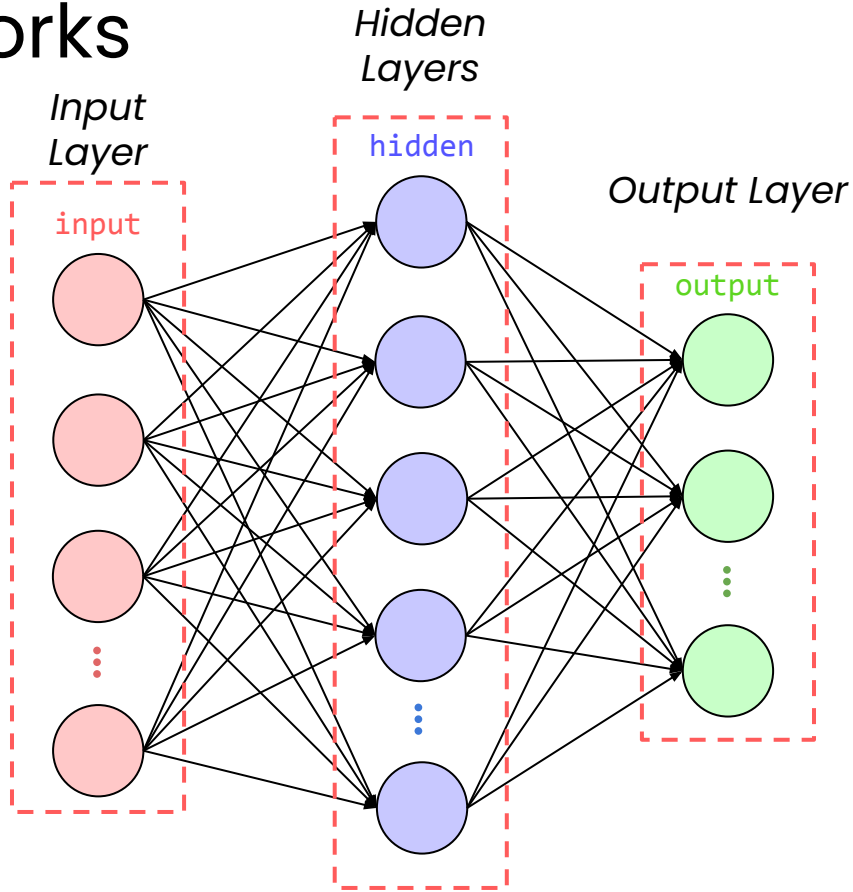
$time = W_1 * distance + b$

Single Neuron



$$time = W_1 * distance + W_2 * time\_of\_day + W_3 * weather + b$$

Neural Networks

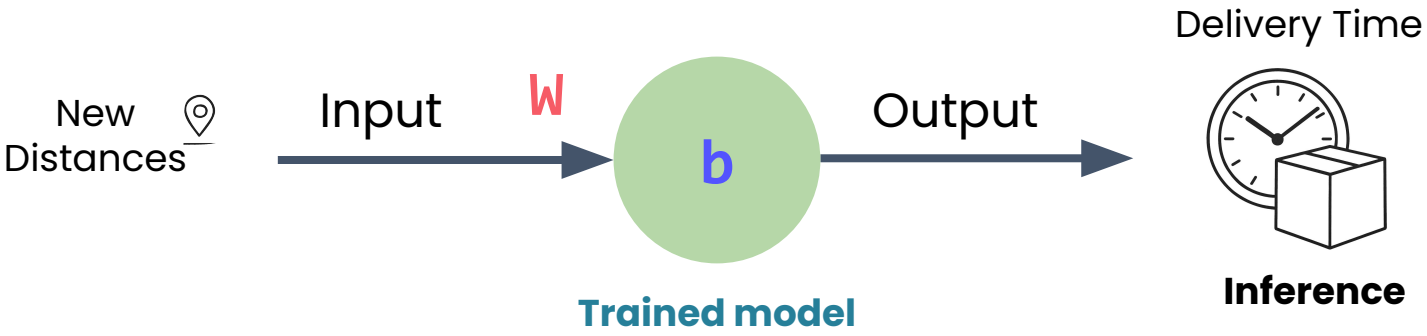


DeepLearning.AI

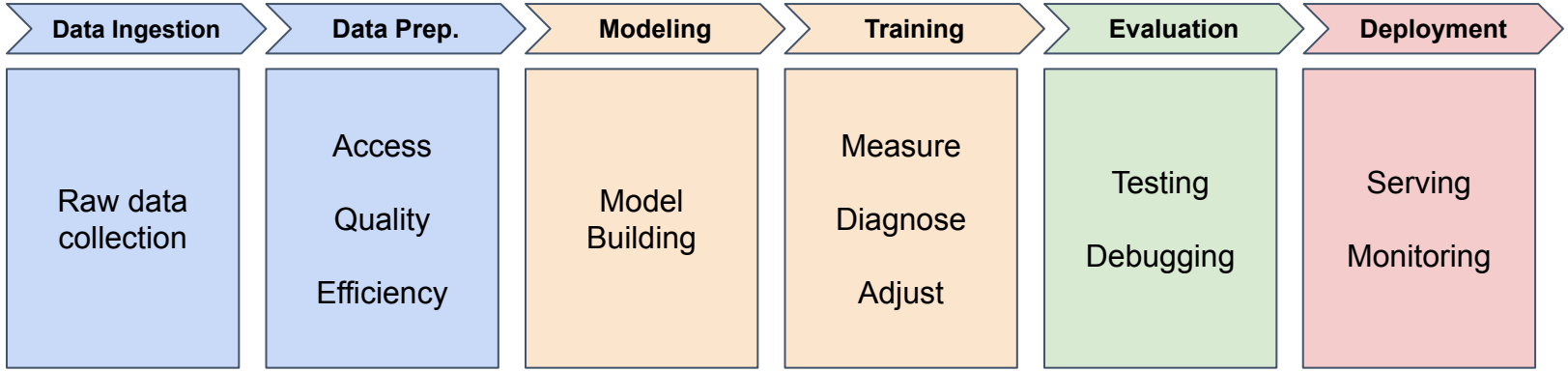
The ML Pipeline

Getting Started with PyTorch

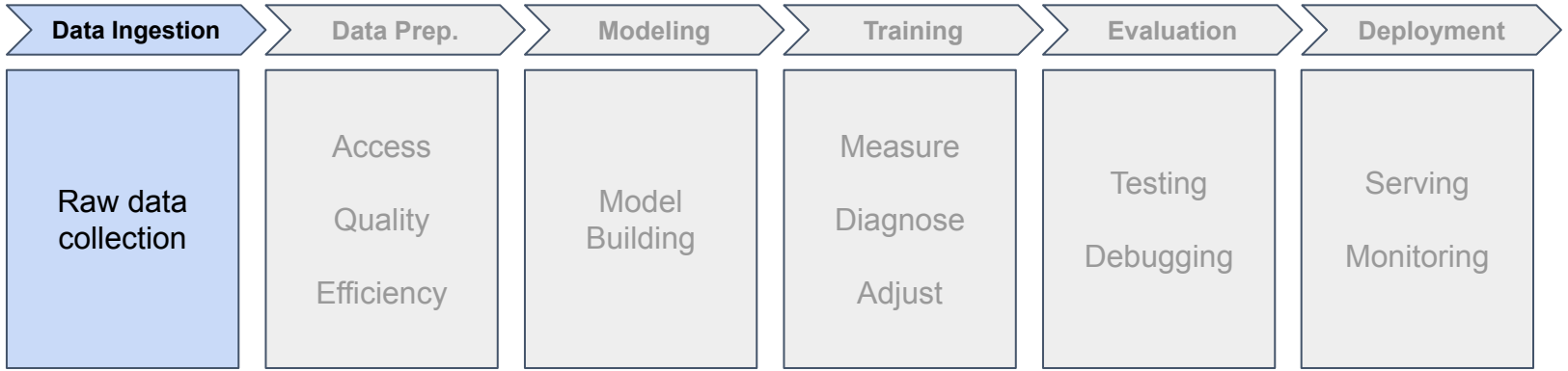
Inference


New
Distances
Input
W
b
Output
Delivery Time
Trained model
Inference

The ML pipeline



Data Ingestion



Data Ingestion

Distance (miles)	Delivery Time (minutes)
2.0	11.9
3.0	15.3
4.0	18.8
5.0	22.2
6.0	25.6

Data Ingestion

Distance (miles)	Delivery Time (minutes)
7	“22 minutes”

Data Ingestion

Distance (miles)	Delivery Time (minutes)
7	“22 minutes”
7	22.0

Data Ingestion

Distance (miles)	Delivery Time (minutes)
7	“22 minutes”
7	22.0
N/A	18.8

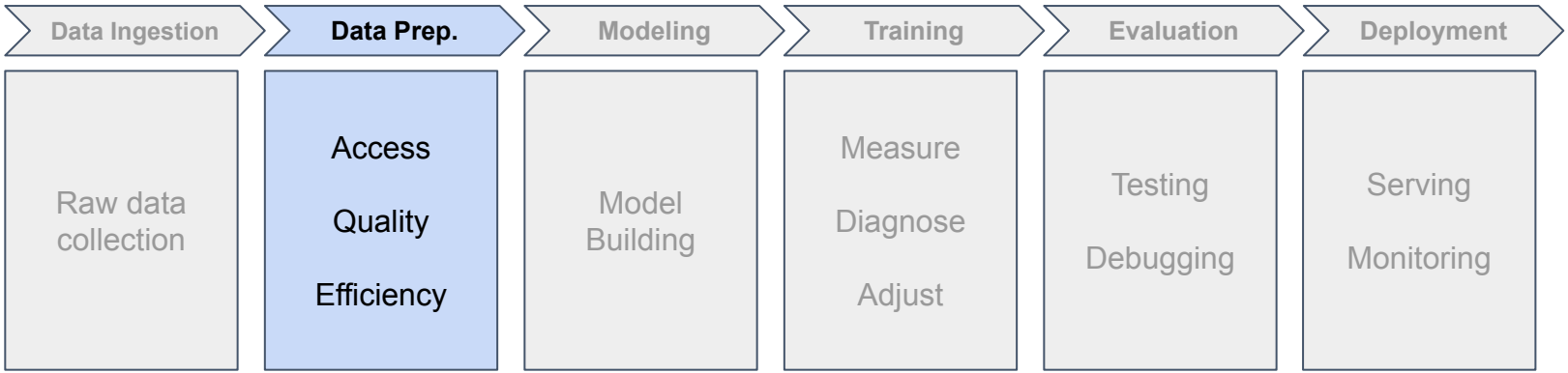
Data Ingestion

Distance (miles)	Delivery Time (minutes)
7	“22 minutes”
7	22.0
N/A	18.8
N/A	-22.2

Data Ingestion

Distance (miles)	Delivery Time (minutes)
7	“22 minutes”
7	22.0
N/A	18.8
N/A	-22.2
10 miles	3 minutes

Data Preparation



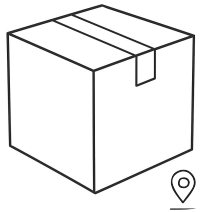
Data Preparation

Sources of data

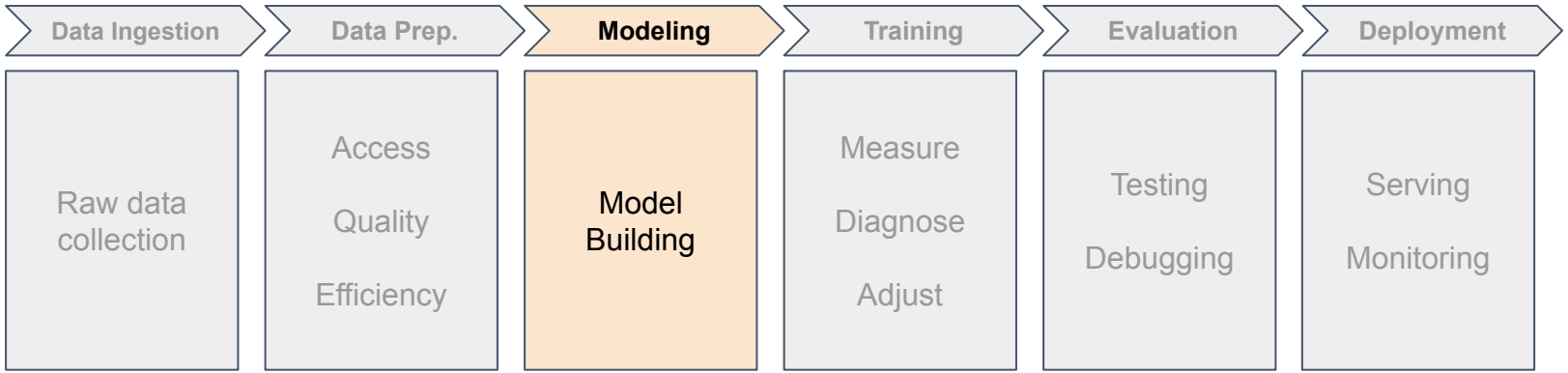
- delivery records
- customer images
- email text
- other

- Cleaning
- Transforming
- Organizing

- Fixing errors
- Handling missing values
- Engineering new features



Model Building



Model Building

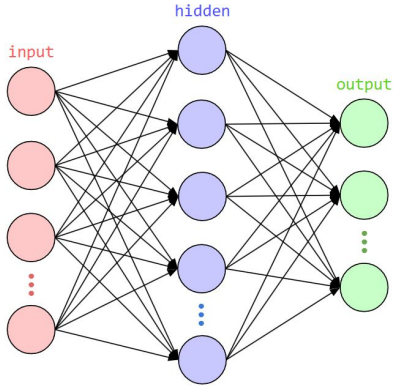
Data Ingestion | Data Prep. | **Modeling** | Training | Evaluation | Deployment

Output

- Delivery times
- Classifying images
- Analyzing text
- Others

Choosing the architecture

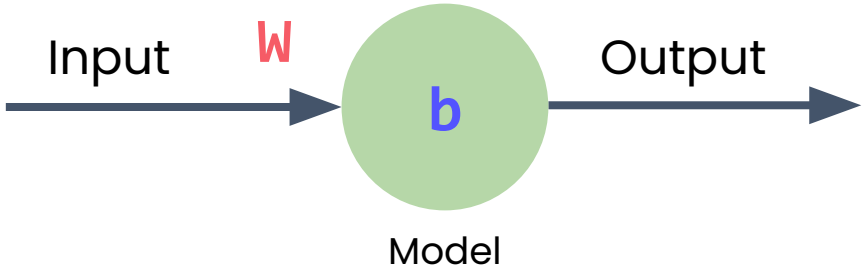
- How many neurons should you use?
- How are they connected?
- What kinds of layers does the model need?
- Cost, time, infrastructure, others?



Final model

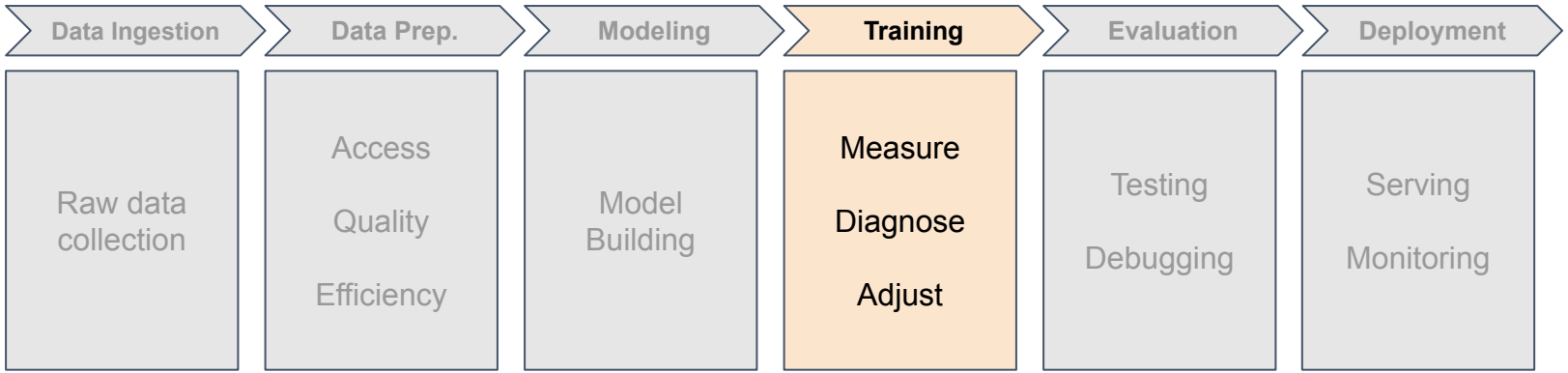
Model Building

Data Ingestion | Data Prep. | **Modeling** | Training | Evaluation | Deployment




```
model = nn.Sequential(nn.Linear(1, 1))
```

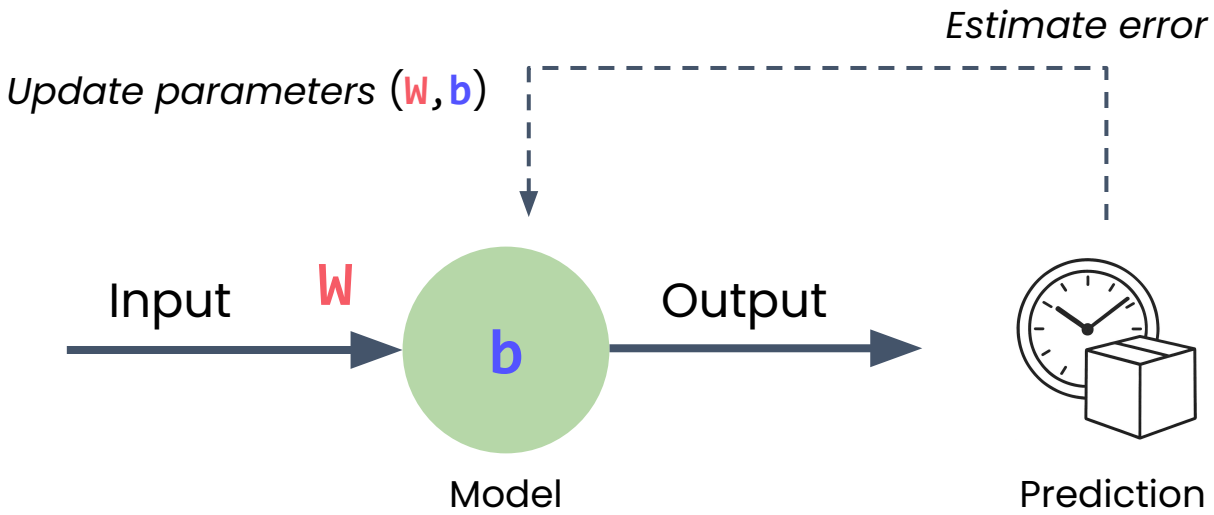
Training


Data Ingestion
Data Prep.
Modeling
Training
Evaluation
Deployment
Raw data collection
Access
Quality
Efficiency
Model Building
Measure
Diagnose
Adjust
Testing
Debugging
Serving
Monitoring

Training

delivery records

- *8.2 miles, 22 minutes*
- *12.5 miles, 31 minutes*

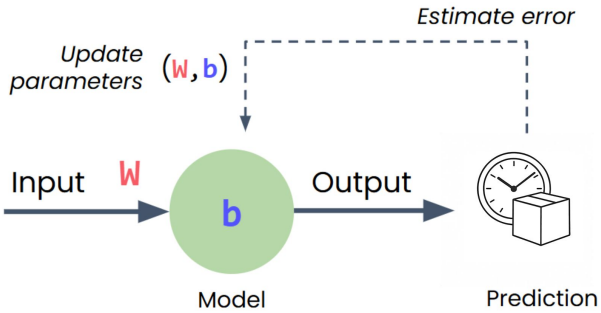


Training

Configure

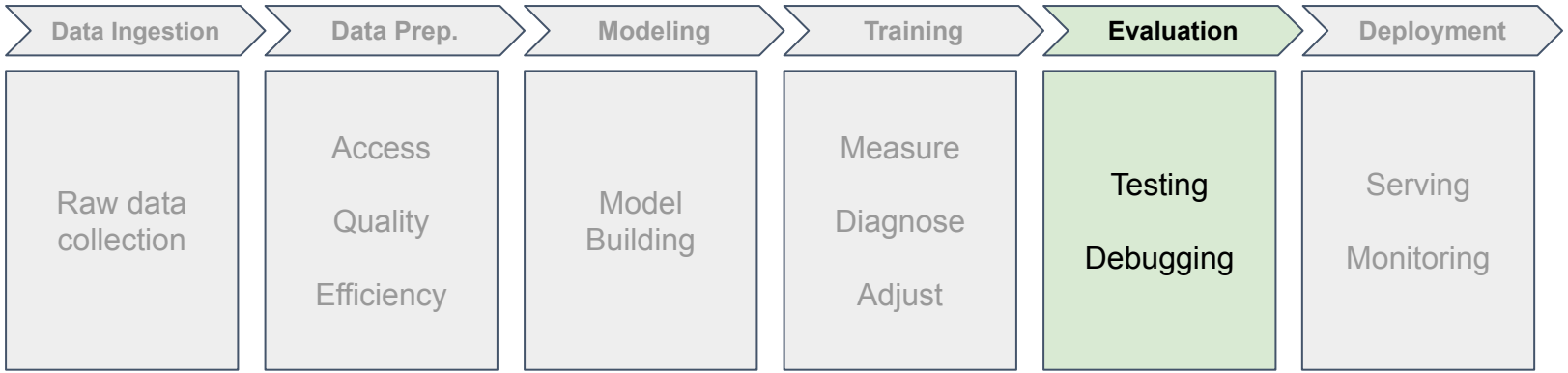
- How to measure prediction errors
- How to guide the model to improve
- How fast it learns

Training Loop



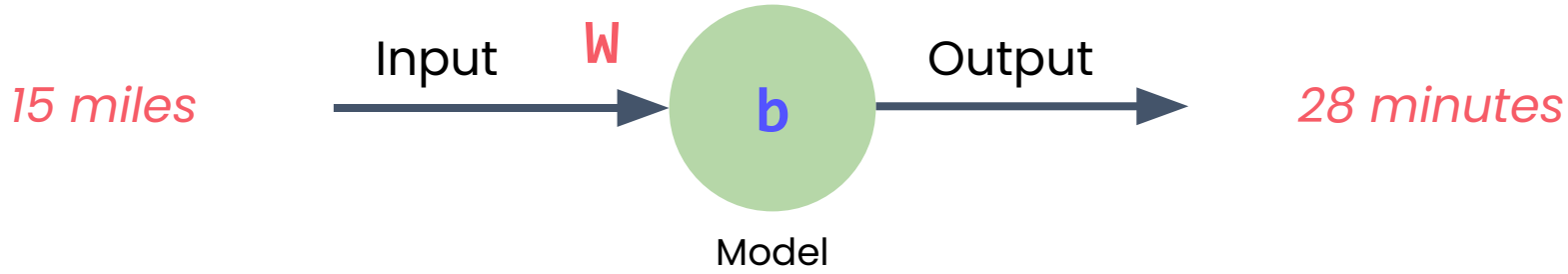
Good predictions?

Evaluation


Data Ingestion
Data Prep.
Modeling
Training
Evaluation
Deployment
Raw data collection
Access
Quality
Efficiency
Model Building
Measure
Diagnose
Adjust
Testing
Debugging
Serving
Monitoring

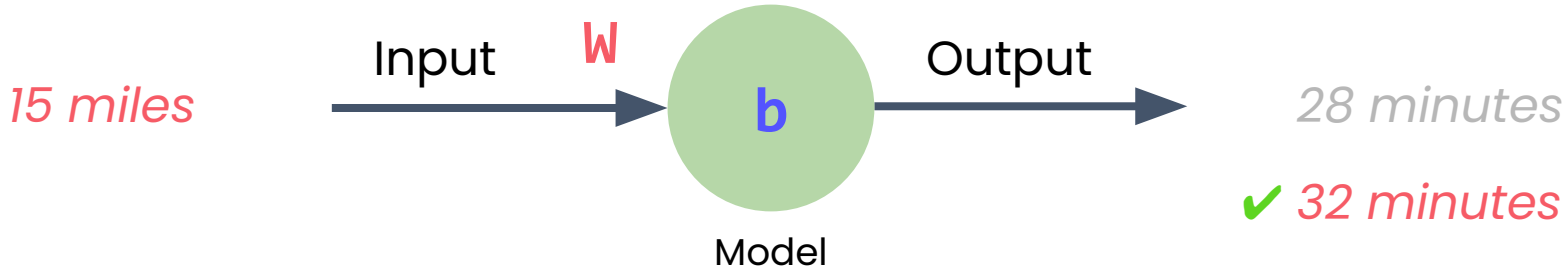
Evaluation & Debugging

Data Ingestion | Data Prep. | Modeling | Training | **Evaluation** | Deployment

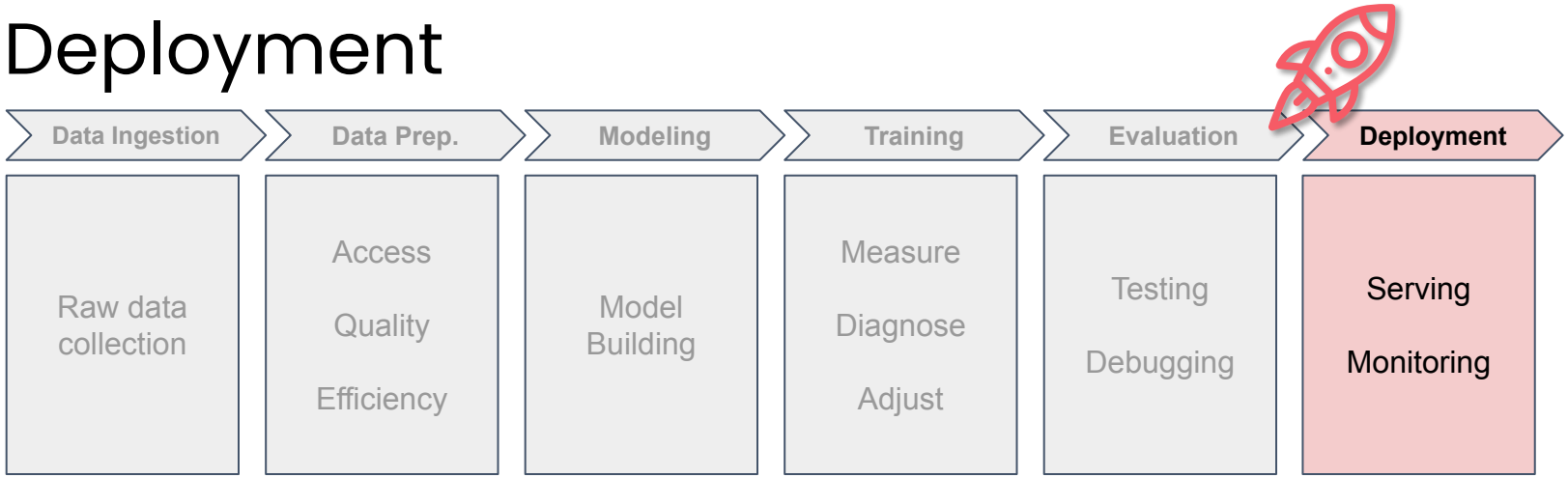


Evaluation & Debugging

Data Ingestion | Data Prep. | Modeling | Training | **Evaluation** | Deployment



Deployment


Data Ingestion
Data Prep.
Modeling
Training
Evaluation
Deployment
Raw data collection
Access
Quality
Efficiency
Model Building
Measure
Diagnose
Adjust
Testing
Debugging
Serving
Monitoring

DeepLearning.AI

Building a Simple Neural Network

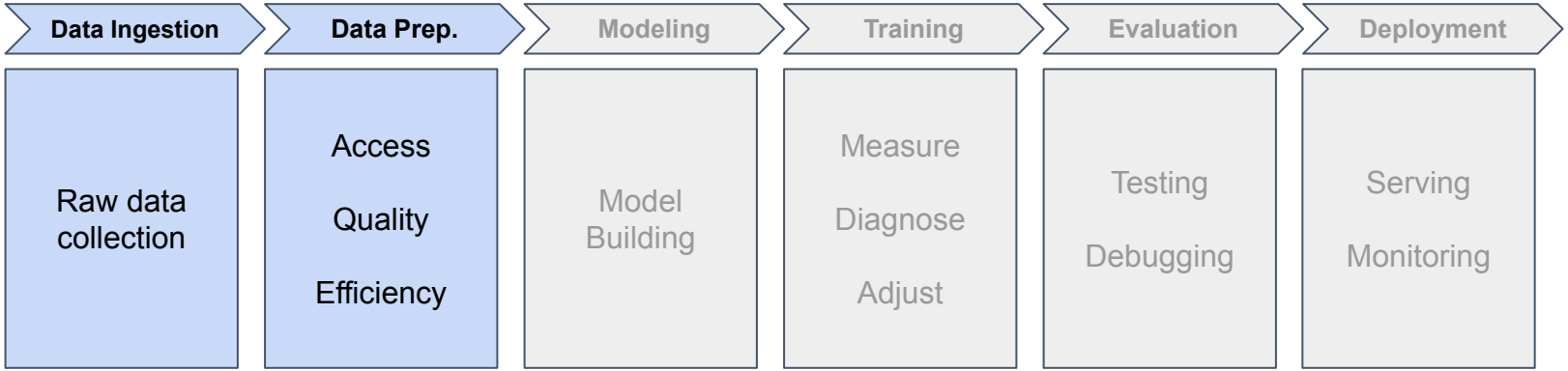
Getting Started with PyTorch

Import Libraries

```
import torch
import torch.nn as nn
import torch.optim as optim
```

- PyTorch's core functionality -> `torch`
- Components for building neural networks -> `nn`
- And tools for training those networks -> `optim`

The ML Pipeline



Defining Data


```
# Distance in miles
distances = torch.tensor([[1.0],[2.0],[3.0],[4.0]], dtype=torch.float32)
# Delivery times in minutes
times = torch.tensor([[6.96],[12.11],[16.77],[22.21]], dtype=torch.float32)
```

Defining Data


```
# Distance in miles
distances = torch.tensor([[1.0],[2.0],[3.0],[4.0]], dtype=torch.float32)
# Delivery times in minutes
times = torch.tensor([[6.96],[12.11],[16.77],[22.21]], dtype=torch.float32)
```

Defining Data


```
# Distance in miles
distances = torch.tensor([[1.0],[2.0],[3.0],[4.0]], dtype=torch.float32)
# Delivery times in minutes
times = torch.tensor([[6.96],[12.11],[16.77],[22.21]], dtype=torch.float32)
```

Defining Data


```
# Distance in miles
distances = torch.tensor([[1.0],[2.0],[3.0],[4.0]], dtype=torch.float32)
# Delivery times in minutes
times = torch.tensor([[6.96],[12.11],[16.77],[22.21]], dtype=torch.float32)
```

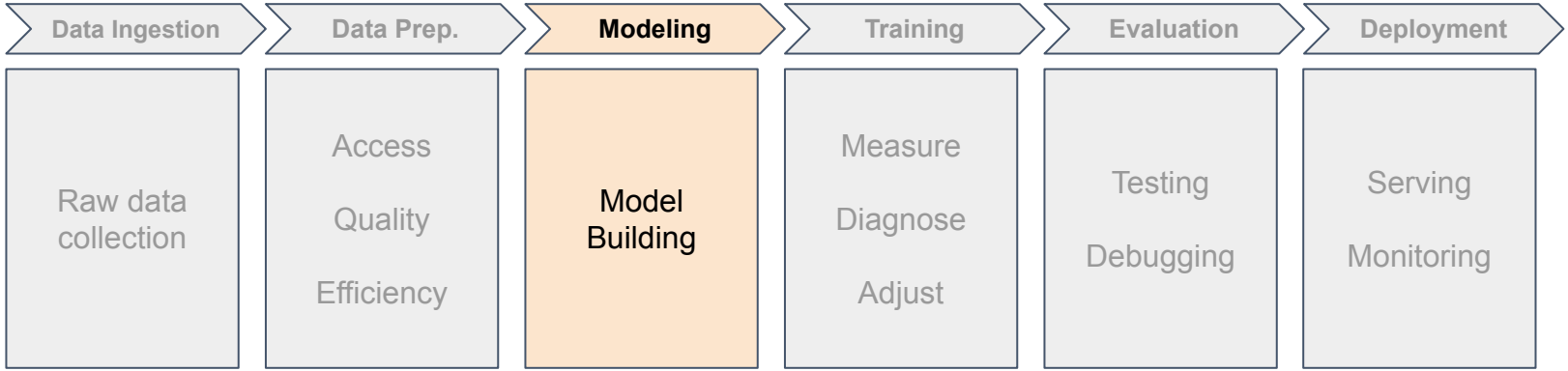
Defining Data - Multiple Inputs


```
# Distance in miles
inputs = torch.tensor([[2.0,3.0,4.0],[6.0,7.0,8.0],[11.0,12.0,13.0],...],dtype=torch.float32)
# Delivery times in minutes
times = torch.tensor([[14.8],[24.1],[36.5],[51.2],[60.8],[75.3]],dtype=torch.float32)
```

Defining Data


```
# Distance in miles
distances = torch.tensor([[1.0],[2.0],[3.0],[4.0]], dtype=torch.float32)
# Delivery times in minutes
times = torch.tensor([[6.96],[12.11],[16.77],[22.21]], dtype=torch.float32)
```

Model Building



Model Building

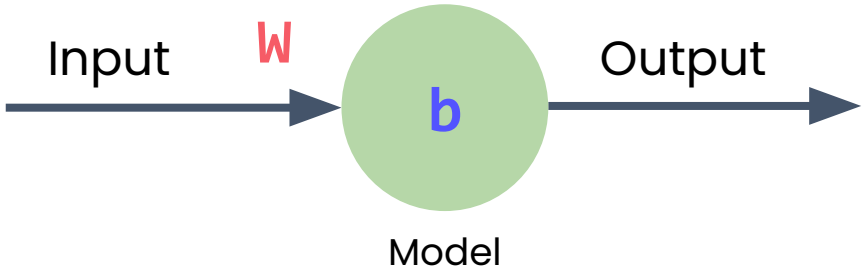

```
# Define the model
model = nn.Sequential(nn.Linear(1, 1))
```

Model Building


```
# Define the model
model = nn.Sequential(nn.Linear(1, 1))
```

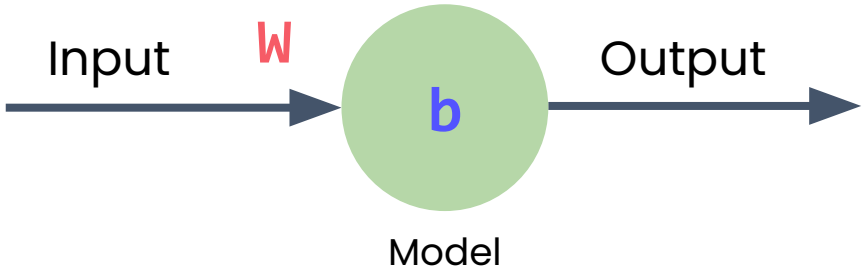
Model Building


```
# Define the model
model = nn.Sequential(nn.Linear(1, 1))
```



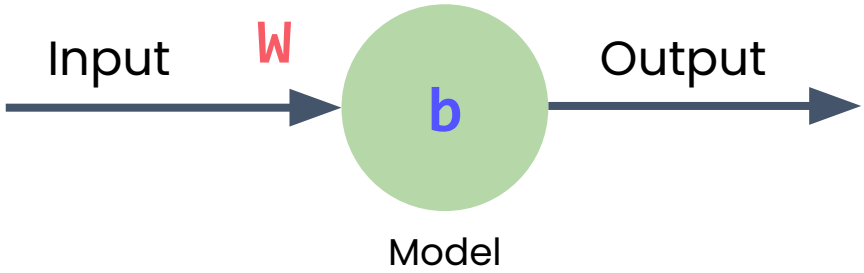
Model Building


```
# Define the model
model = nn.Sequential(nn.Linear(1, 1))
```



Model Building

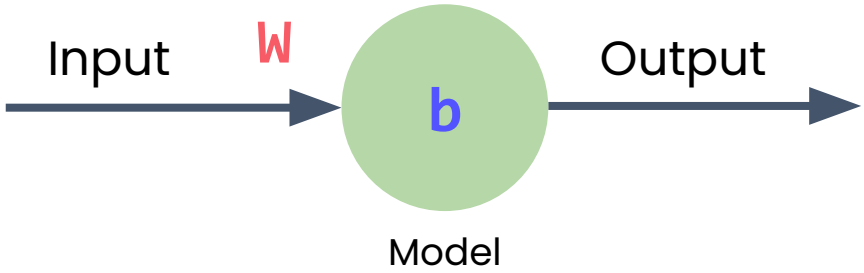

```
# Define the model
model = nn.Sequential(nn.Linear(1, 1))
```



Model Building

Data Ingestion | Data Prep. | **Modeling** | Training | Evaluation | Deployment

```
# Define the model
model = nn.Sequential(nn.Linear(1, 1)
```

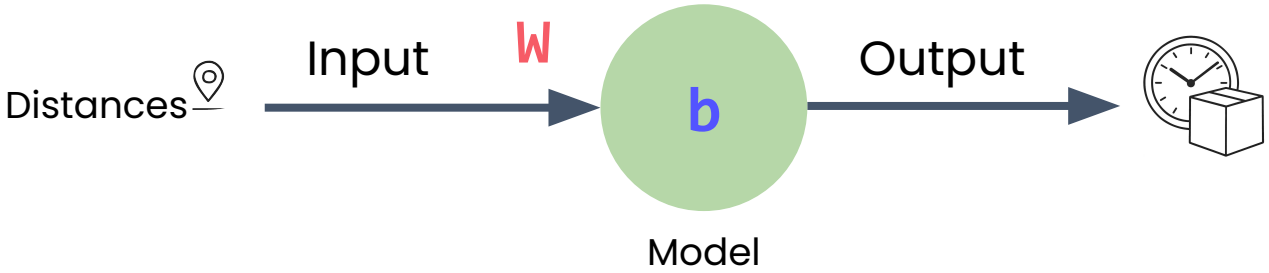


Output = W * Input + b

Model Building

Data Ingestion | Data Prep. | **Modeling** | Training | Evaluation | Deployment

```
# Define the model
model = nn.Sequential(nn.Linear(1, 1))
```



Output = W * Input + b

Loss Function & Optimizer


```
# Define the loss function and optimizer
loss_function = nn.MSELoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)
```

Loss Function & Optimizer


```
# Define the loss function and optimizer
loss_function = nn.MSELoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)
```

Mean Squared Error Loss

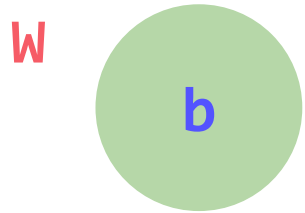
Loss Function & Optimizer


```
# Define the loss function and optimizer
loss_function = nn.MSELoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)
```

Stochastic Gradient Descent

Loss Function & Optimizer


```
# Define the loss function and optimizer
loss_function = nn.MSELoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)
```

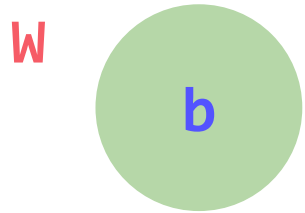


*time= W * distance + b*

Loss Function & Optimizer

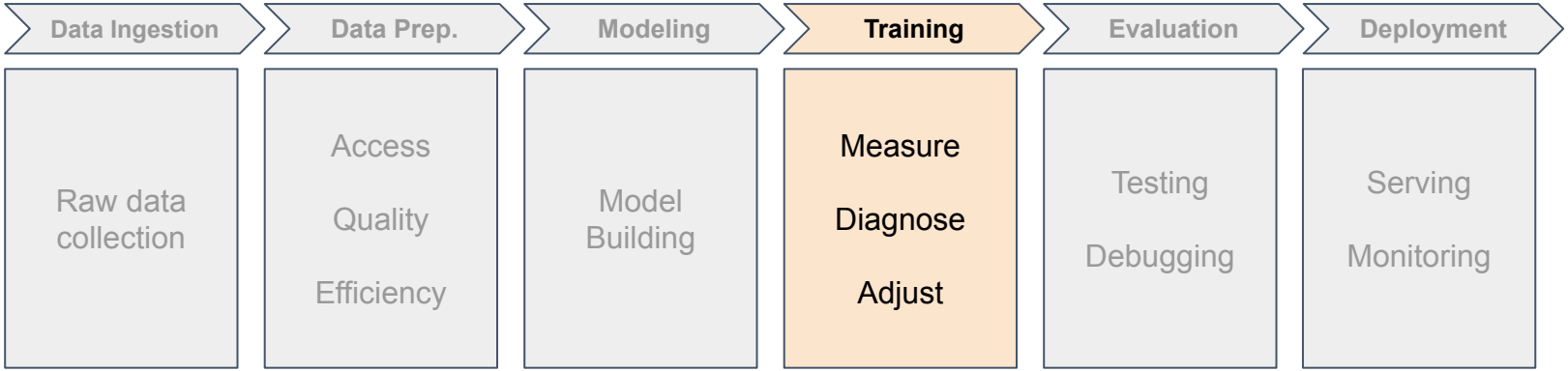

```
# Define the loss function and optimizer
loss_function = nn.MSELoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)
```

Learning Rate



*time= W * distance + b*

Training


Data Ingestion
Data Prep.
Modeling
Training
Evaluation
Deployment
Raw data collection
Access
Quality
Efficiency
Model Building
Measure
Diagnose
Adjust
Testing
Debugging
Serving
Monitoring

Training


```
# Training loop
for epoch in range(500):
    # 0. Reset the optimizer
    optimizer.zero_grad()
    # 1. Make predictions
    outputs = model(inputs)
    # 2. Calculate the loss - how bad was this guess?
    loss = loss_function(outputs, times)
    # 3. Calculate adjustments
    loss.backward()
    # 4. Update the model
    optimizer.step()
```

Training


```
# Training loop
for epoch in range(500):
    # 0. Reset the optimizer
    optimizer.zero_grad()
    # 1. Make predictions
    outputs = model(inputs)
    # 2. Calculate the loss - how bad was this guess?
    loss = loss_function(outputs, times)
    # 3. Calculate adjustments
    loss.backward()
    # 4. Update the model
    optimizer.step()
```

Training


```
# Training loop
for epoch in range(500):
    # 0. Reset the optimizer
    optimizer.zero_grad()
    # 1. Make predictions
    outputs = model(inputs)
    # 2. Calculate the loss - how bad was this guess?
    loss = loss_function(outputs, times)
    # 3. Calculate adjustments
    loss.backward()
    # 4. Update the model
    optimizer.step()
```

Training


```
# Training loop
for epoch in range(500):
    # 0. Reset the optimizer
    optimizer.zero_grad()
    # 1. Make predictions
    outputs = model(inputs)
    # 2. Calculate the loss - how bad was this guess?
    loss = loss_function(outputs, times)
    # 3. Calculate adjustments
    loss.backward()
    # 4. Update the model
    optimizer.step()
```

Training


```
# Training loop
for epoch in range(500):
    # 0. Reset the optimizer
    optimizer.zero_grad()
    # 1. Make predictions
    outputs = model(inputs)
    # 2. Calculate the loss - how bad was this guess?
    loss = loss_function(outputs, times)
    # 3. Calculate adjustments
    loss.backward()
    # 4. Update the model
    optimizer.step()
```

Training


```
# Training loop
for epoch in range(500):
    # 0. Reset the optimizer
    optimizer.zero_grad()
    # 1. Make predictions
    outputs = model(inputs)
    # 2. Calculate the loss - how bad was this guess?
    loss = loss_function(outputs, times)
    # 3. Calculate adjustments
    loss.backward()
    # 4. Update the model
    optimizer.step()
```

Training


```
# Training loop
for epoch in range(500):
    # 0. Reset the optimizer
    optimizer.zero_grad()
    # 1. Make predictions
    outputs = model(inputs)
    # 2. Calculate the loss - how bad was this guess?
    loss = loss_function(outputs, times)
    # 3. Calculate adjustments
    loss.backward()
    # 4. Update the model
    optimizer.step()
```

Training


```
# Training loop
for epoch in range(500):
    # 0. Reset the optimizer
    optimizer.zero_grad()
    # 1. Make predictions
    outputs = model(inputs)
    # 2. Calculate the loss - how bad was this guess?
    loss = loss_function(outputs, times)
    # 3. Calculate adjustments
    loss.backward()
    # 4. Update the model
    optimizer.step()
```

Training


```
# Training loop
for epoch in range(500):
    # 0. Reset the optimizer
    optimizer.zero_grad()
    # 1. Make predictions
    outputs = model(inputs)
    # 2. Calculate the loss - how bad was this guess?
    loss = loss_function(outputs, times)
    # 3. Calculate adjustments
    loss.backward()
    # 4. Update the model
    optimizer.step()
```

Back Propagation

Training


```
# Training loop
for epoch in range(500):
    # 0. Reset the optimizer
    optimizer.zero_grad()
    # 1. Make predictions
    outputs = model(inputs)
    # 2. Calculate the loss - how bad was this guess?
    loss = loss_function(outputs, times)
    # 3. Calculate adjustments
    loss.backward()
    # 4. Update the model
    optimizer.step()
```

Inference

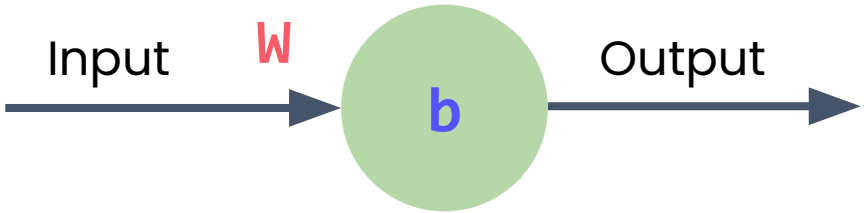
```
with torch.no_grad():
    test_distance = torch.tensor([[25.0]], dtype=torch.float32)
    predicted_time = model(test_distance)
    print(f"Predicted time for 25 miles: {predicted_time.item():.1f} minutes")
```

Inference

```
with torch.no_grad():
    test_distance = torch.tensor([[25.0]], dtype=torch.float32)
    predicted_time = model(test_distance)
    print(f"Predicted time for 25 miles: {predicted_time.item():.1f} minutes")
```

Inference

```
with torch.no_grad():
    test_distance = torch.tensor([[25.0]], dtype=torch.float32)
    predicted_time = model(test_distance)
    print(f"Predicted time for 25 miles: {predicted_time.item():.1f} minutes")
```



DeepLearning.AI

Activation Functions

Getting Started with PyTorch

The Delivery Problem

The Delivery Problem

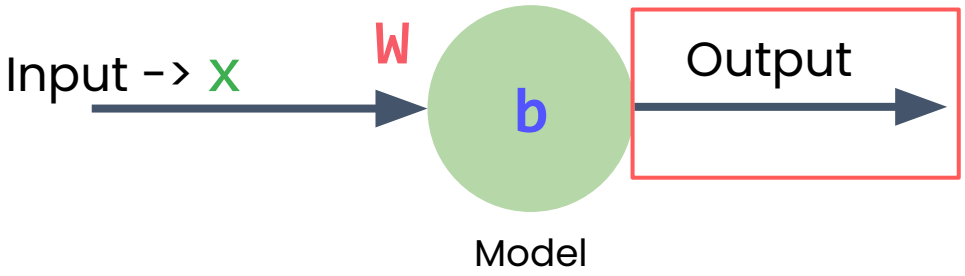
The Delivery Problem

The Delivery Problem

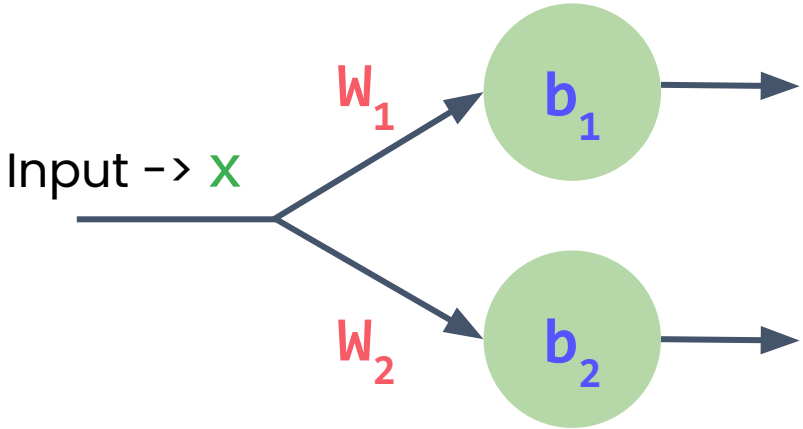
The Delivery Problem

The Delivery Problem

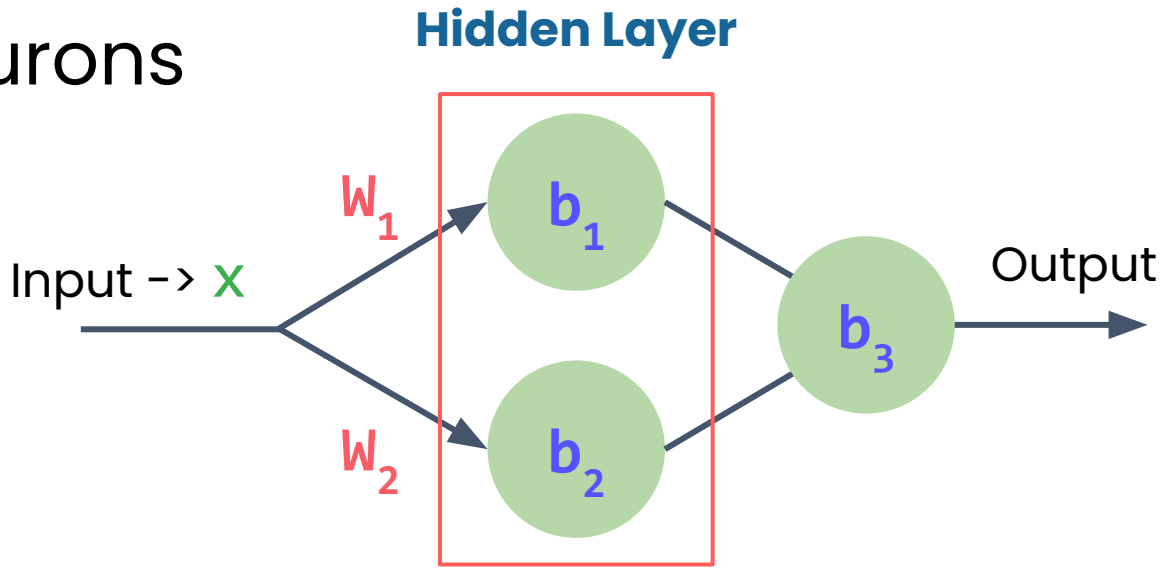
Single Neurons



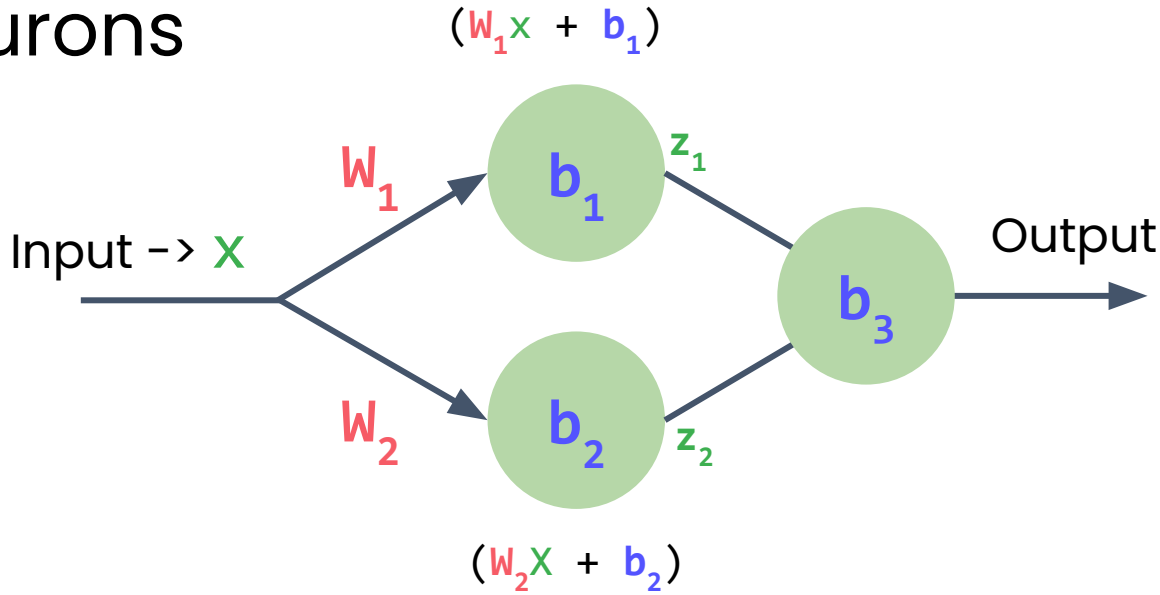
Two Neurons



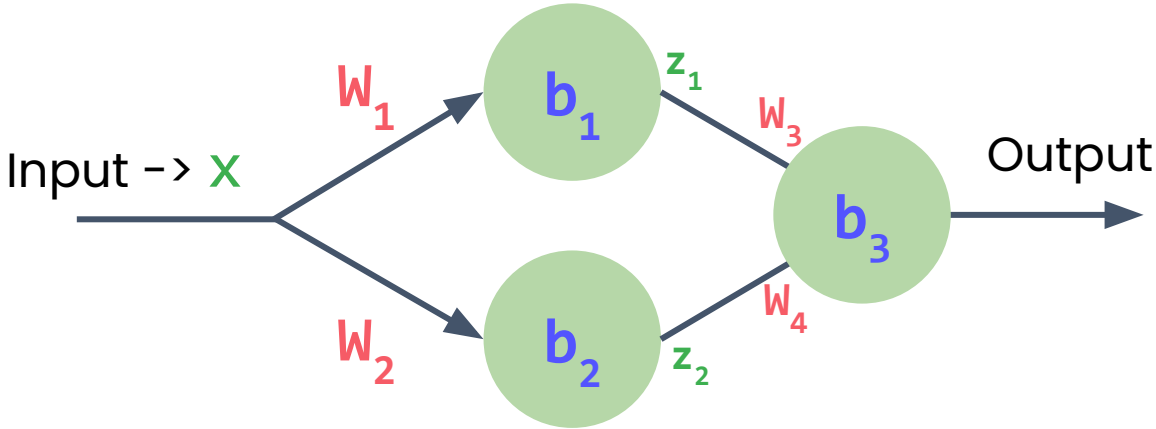
Two Neurons



Two Neurons



Two Neurons



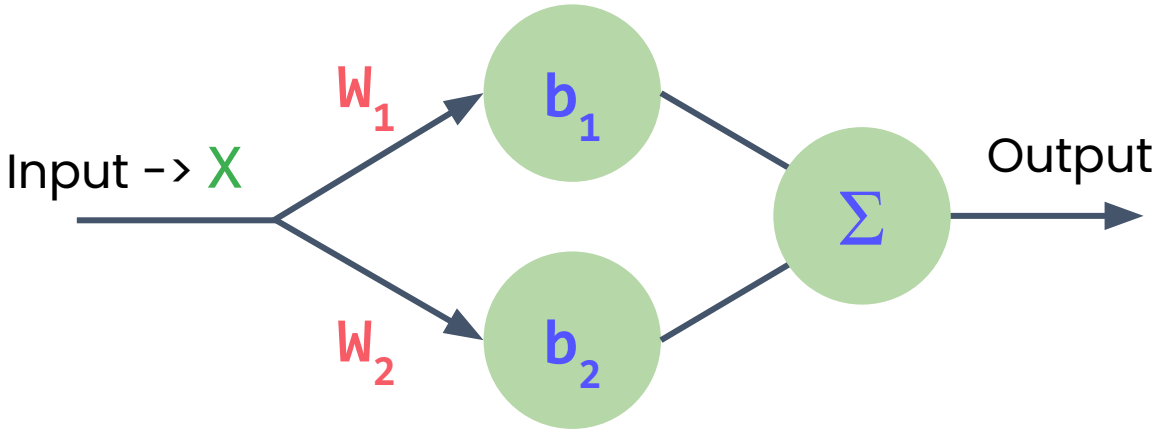
$$\text{Output} = W_3(W_1x+b_1) + W_4(W_2x+b_2) + b_3$$

$$(W_3W_1+W_4W_2)x + (W_3b_1+W_4b_2+b_3)$$

$$W*x + b$$

Still a linear equation!

Two Neurons



$$\text{Output} = (W_1 X + b_1) + (W_2 X + b_2)$$

$$(W_1 + W_2) * X + (b_1 + b_2)$$

$$W * X + b$$

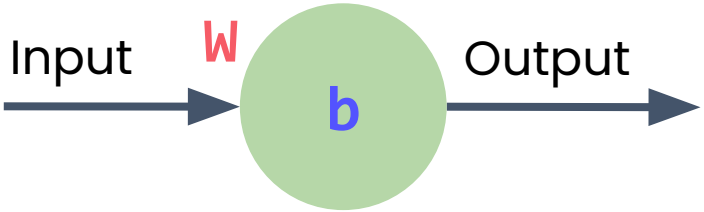
Still a linear equation!

The Delivery Problem

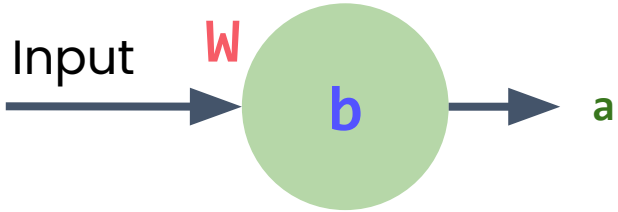
The Delivery Problem

W*X + b

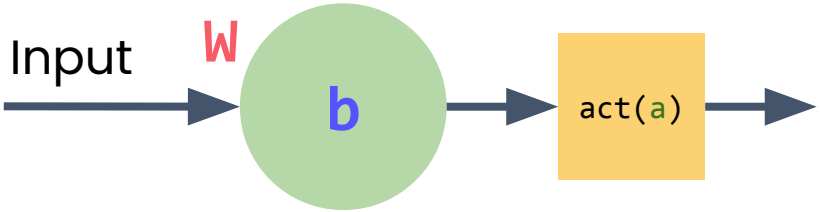
Activation Functions



Activation Functions



Activation Functions

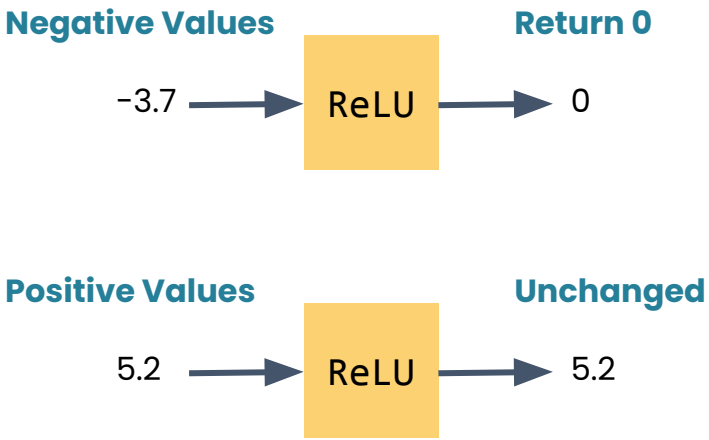


$$a = W*X + b$$

$$Output = act(a)$$

ReLU

```
nn.ReLU()
```




```
if input > 0:
    return input
else:
    return 0
```

ReLU

```
# Define the model
model = nn.Sequential(
    nn.Linear(1, 1),
    nn.ReLU()
)
```

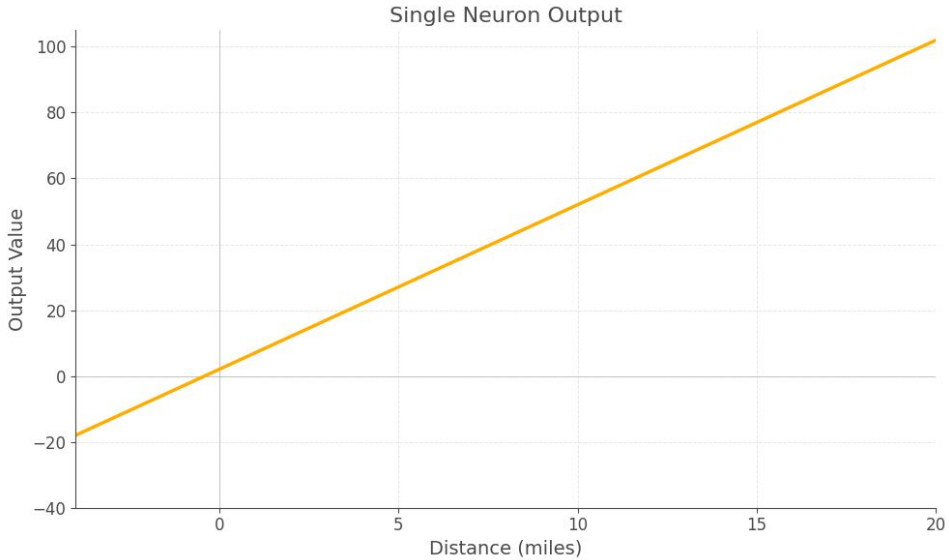
ReLU

```
# Define the model
model = nn.Sequential(
    nn.Linear(1, 1),
    nn.ReLU()
)
```

ReLU

```
# Define the model
model = nn.Sequential(
    nn.Linear(1, 1),
    nn.ReLU()
)
```

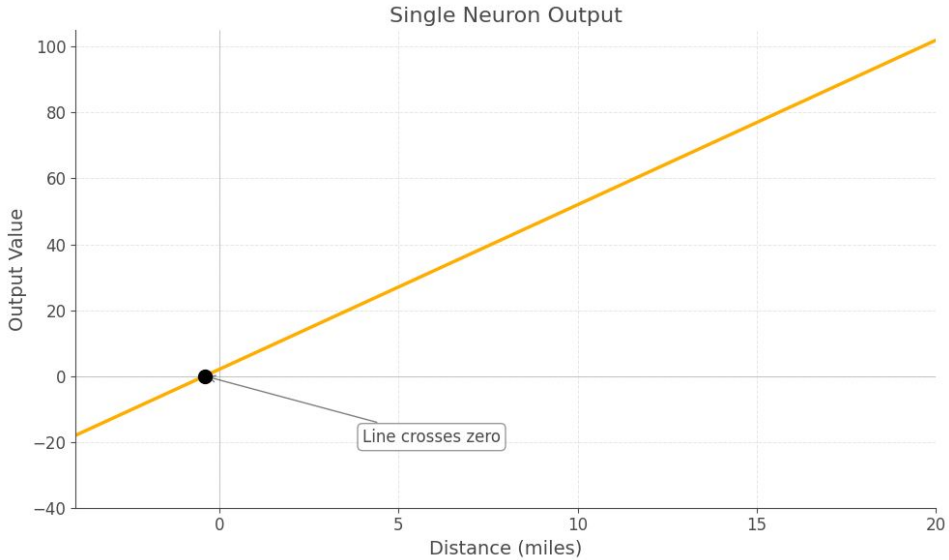
ReLU



$$time = W * distance + b$$

$$\text{ReLU}(time)$$

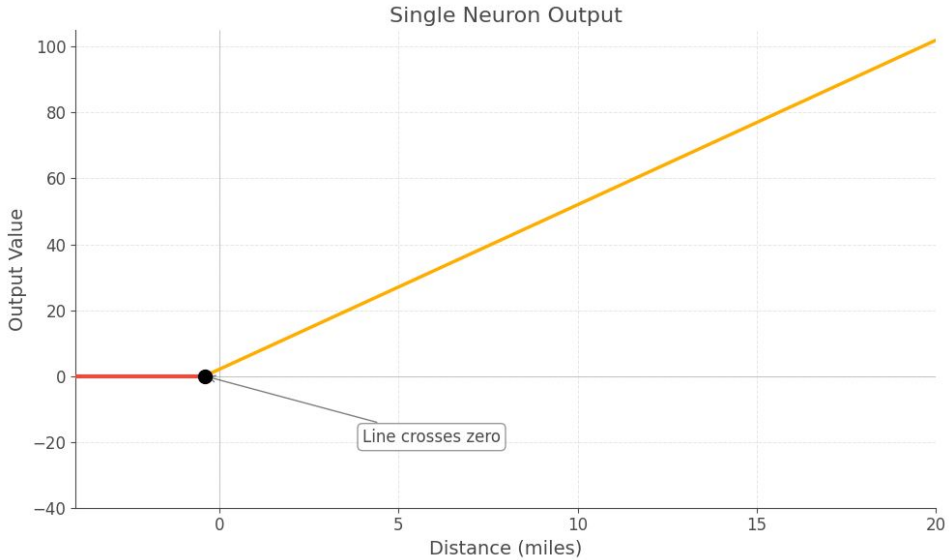
ReLU



$$time = W * distance + b$$

$$\text{ReLU}(time)$$

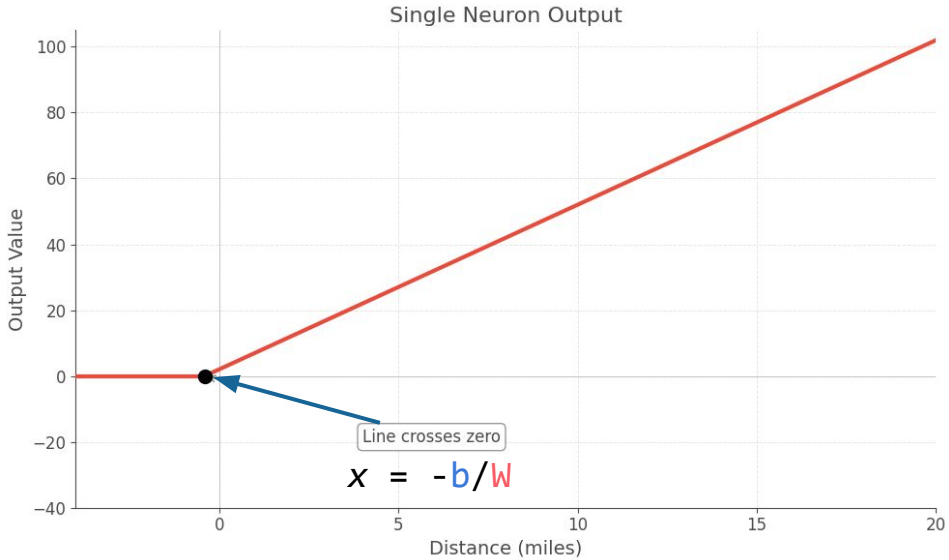
ReLU



$$time = W * distance + b$$

$$\text{ReLU}(time)$$

ReLU



$$time = W * distance + b$$

$$\text{ReLU}(time)$$

The Delivery Problem

The Delivery Problem

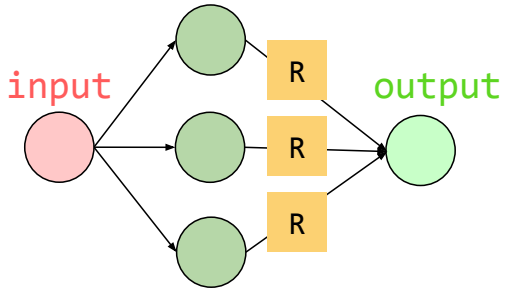
ReLU

W_1 b_1 W_2 b_2 W_3 b_3

$-b_1/W_1$ $-b_2/W_2$ $-b_3/W_3$

ReLU


Delivery Time vs Distance
Delivery Time (minutes)
0
10
20
30
40
50
60
70
80
90
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Distance (miles)

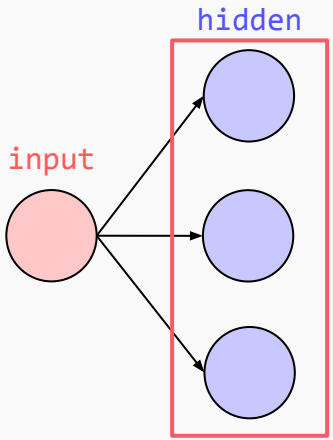

input
R
R
R
output

Non Linear Model

```
model = nn.Sequential(
    nn.Linear(1, 3),
    nn.ReLU(),
    nn.Linear(3, 1)
)
```

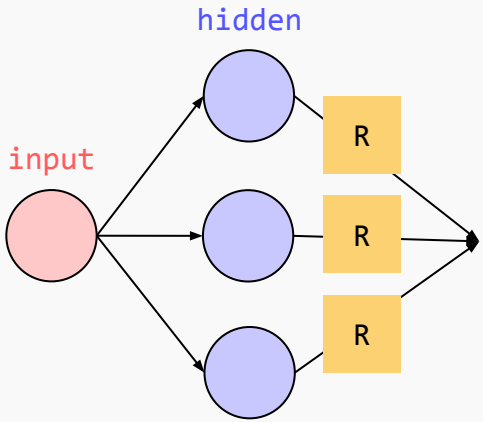
Non Linear Model

```
model = nn.Sequential(
    nn.Linear(1, 3),
    nn.ReLU(),
    nn.Linear(3, 1)
)
```



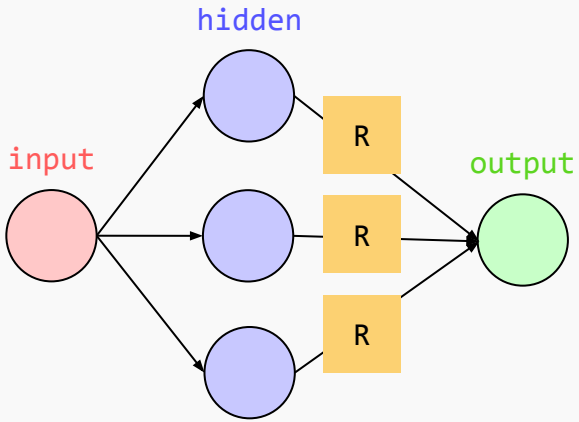
Non Linear Model

```
model = nn.Sequential(
    nn.Linear(1, 3),
    nn.ReLU(),
    nn.Linear(3, 1)
)
```



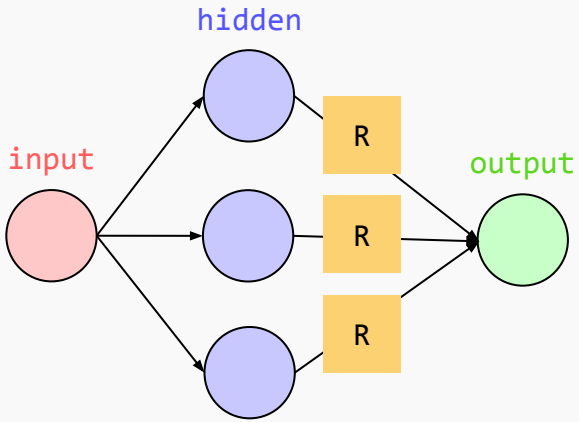
Non Linear Model

```
model = nn.Sequential(
    nn.Linear(1, 3),
    nn.ReLU(),
    nn.Linear(3, 1)
)
```



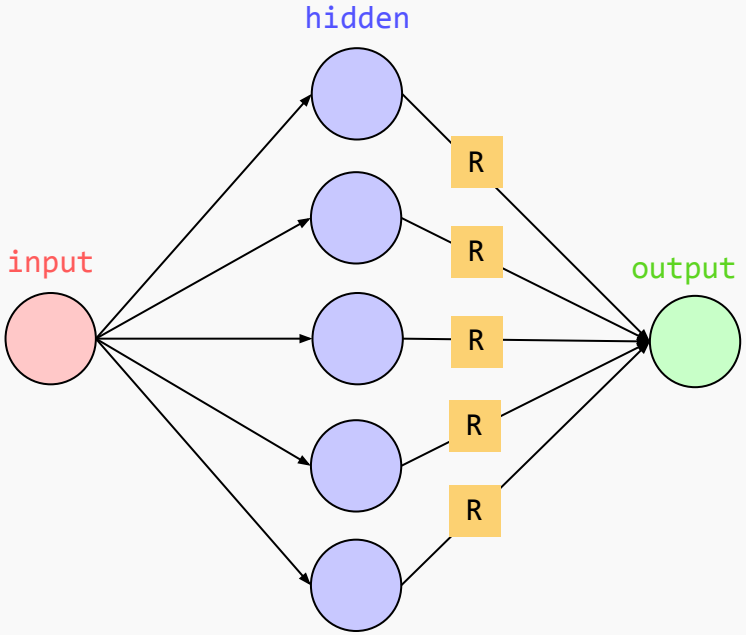
Non Linear Model

```
model = nn.Sequential(
    nn.Linear(1, 3),
    nn.ReLU(),
    nn.Linear(3, 1)
)
```

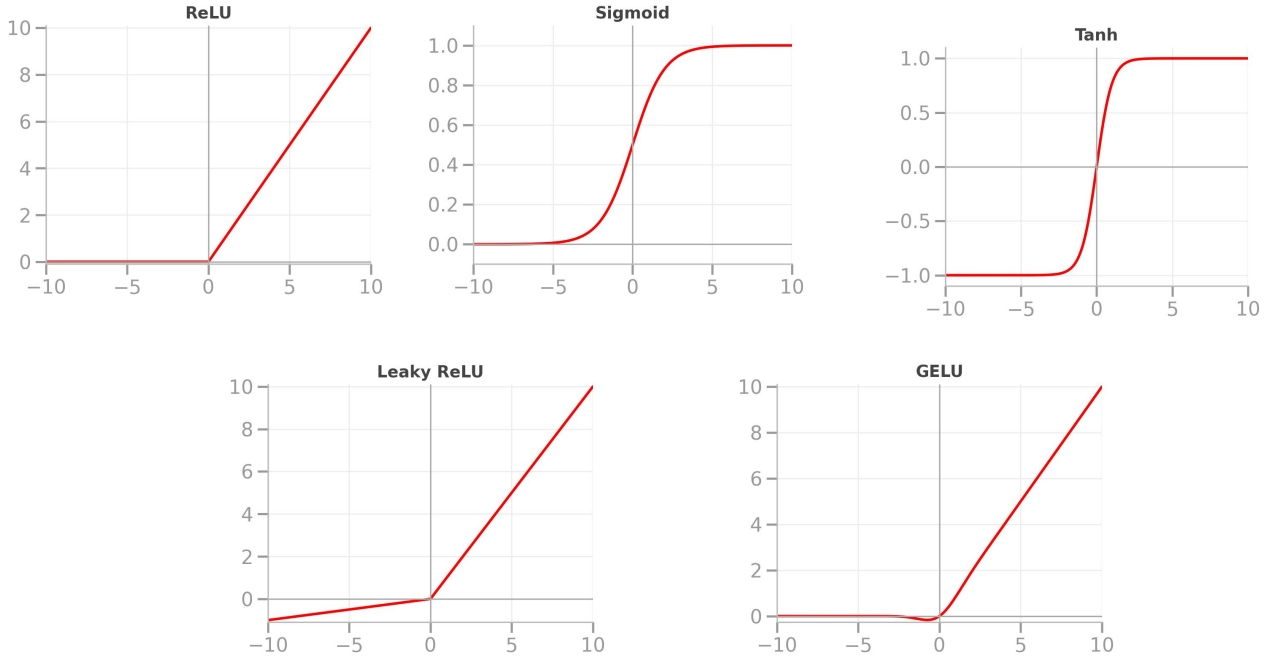


Non Linear Model

```
model = nn.Sequential(
    nn.Linear(1, 5),
    nn.ReLU(),
    nn.Linear(5, 1)
)
```



Activation Functions



https://docs.pytorch.org/

DeepLearning.AI

Specialization

Deep Learning Specialization

Build neural networks (CNNs, RNNs, LSTMs, Transformers) and apply them to speech recognition, NLP, and mor...

DeepLearning.AI

Specialization

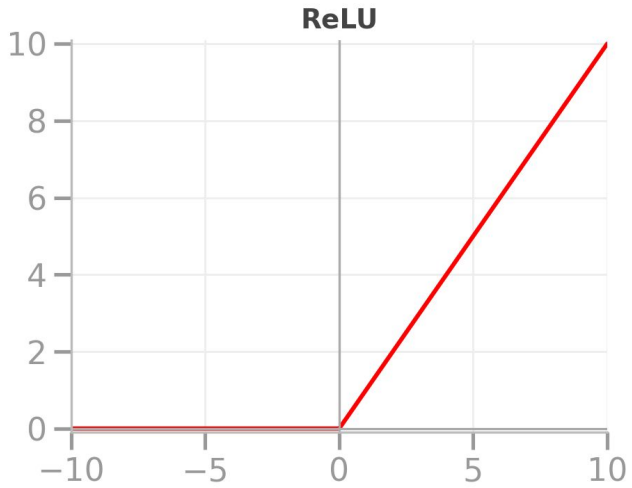
Machine Learning Specialization

Learn foundational AI concepts through an intuitive visual approach, then learn the code needed to...

DeepLearning.AI, Stanford Online

https://www.deeplearning.ai/courses/deep-learning-specialization/

https://www.deeplearning.ai/courses/machine-learning-specialization/


ReLU
10
8
6
4
2
0
−10
−5
0
5
10

DeepLearning.AI

Tensors

Getting Started with PyTorch

You will learn how to...

- Read and understand tensor shapes
- Handle data types
- Create tensors
- Reshape and slice

Tensor Shapes

```
# Your delivery data
distances = torch.tensor([[3.0], [7.0], [12.0], [18.0], [22.0], [28.0]])
print(distances.shape)
```

Output:

```
C:>
torch.Size([6, 1])
```

Tensor Shapes

```
# Your delivery data
distances = torch.tensor([[3.0], [7.0], [12.0], [18.0], [22.0], [28.0]])
print(distances.shape)
```

Output:

```
C:>
torch.Size([6, 1])
```

Tensor Shapes

```
# Your delivery data
distances = torch.tensor([[3.0], [7.0], [12.0], [18.0], [22.0], [28.0]])
print(distances.shape)
```

Output:

```
C:>
torch.Size([6, 1])
```

features per sample

Tensor Shapes

```
# Your delivery data
distances = torch.tensor([[3.0], [7.0], [12.0], [18.0], [22.0], [28.0]])
print(distances.shape)
simple_model = nn.Linear(1, 1)  # Expects 1 feature per sample
```

Output:

```
C:>
torch.Size([6, 1])
```

Tensor Shapes

```
# Your delivery data
distances = torch.tensor([[3.0], [7.0], [12.0], [18.0], [22.0], [28.0]])
print(distances.shape)
simple_model = nn.Linear(1, 1)  # Expects 1 feature per sample

output = simple_model(distances) # Works!
```

Output:

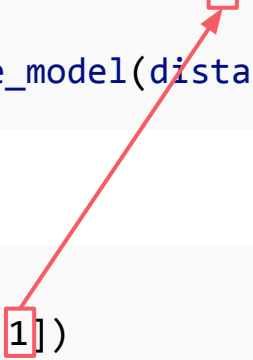
```
C:>
torch.Size([6, 1])
```

Tensor Shapes

```
# Your delivery data
distances = torch.tensor([[3.0], [7.0], [12.0], [18.0], [22.0], [28.0]])
print(distances.shape)
simple_model = nn.Linear(1, 1)  # Expects 1 feature per sample

output = simple_model(distances) # Works!
```

Output:

```
C:>
torch.Size([6, 1])
```



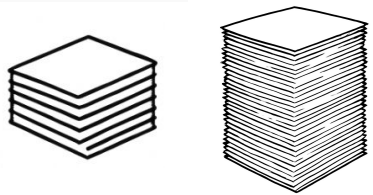
Tensor Shapes

```
# Your delivery data
distances = torch.tensor([[3.0], [7.0], [12.0], [18.0], [22.0], [28.0]])
print(distances.shape)
simple_model = nn.Linear(1, 1)  # Expects 1 feature per sample

output = simple_model(distances)
```

```
C:>
torch.Size([6, 1])
```

?

batch size



Tensor Shapes

```
# Your delivery data
distances = torch.tensor([[3.0], [7.0], [12.0], [18.0], [22.0], [28.0]])
print(distances.shape)
simple_model = nn.Linear(1, 1)  # Expects 1 feature per sample

output = simple_model(distances)
```

```
C:>
torch.Size([6, 1])
```

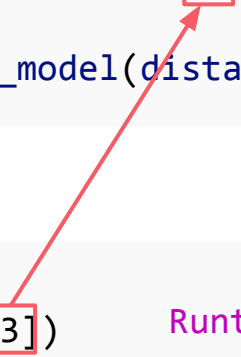
Tensor Shapes


```
# Your delivery data
distances = torch.tensor([[3.0, 7.0, 1.0], [18.0, 22.0, 2.0]])
print(distances.shape)
simple_model = nn.Linear(1, 1)

output = simple_model(distances)
```

```
C:>
torch.Size([2, 3])
```

Tensor Shapes

```
# Your delivery data
distances = torch.tensor([[3.0, 7.0, 1.0], [18.0, 22.0, 2.0]])
print(distances.shape)
simple_model = nn.Linear(1, 1)

output = simple_model(distances)
```



```
C:>
torch.Size([2, 3])      RuntimeError
```

Tensor Shape Errors

```
RuntimeError: mat1 and mat2 shapes cannot be multiplied (2x3 and 1x1)
```

PyTorch will tell you what went wrong, but not always how to fix it.

Tensor Data Types

```
int_tensor = torch.tensor([1, 2, 3])
print(int_tensor.dtype)
```

Output:

```
C:>
torch.int64
```

Tensor Data Types

```
int_tensor = torch.tensor([1, 2, 3])
print(int_tensor.dtype)
```

Output:

```
C:>
torch.int64
```

```
float_tensor = torch.tensor([1.0, 2.0, 3.0])
print(float_tensor.dtype)
```

Output:

```
C:>
torch.float32
```

Tensor Data Types

```
float_tensor = torch.tensor([1, 2, 3], dtype=torch.float32)
```

Tensor Data Types

```
float_tensor = torch.tensor([1, 2, 3], dtype=torch.float32)

float_tensor = int_tensor.float()  # Convert to float32
```

Mixing Types

```
float_tensor = torch.tensor([1.0, 2.0, 3.0], dtype=torch.float32)

int_tensor = torch.tensor([1, 2, 3], dtype=torch.int64)

mixed_tensor = float_tensor + int_tensor

print(f"mixed type: {mixed_tensor.dtype}")
```

```
> mixed type: torch.float32
```

Tensor Data Types

- `float64` -> extra precision
- `int8` -> memory efficiency
- `float32` -> is often the default and works reliably

Creating Tensors

```
my_distances = [[3.0], [7.0], [12.0], [18.0]]
tensor_from_list = torch.tensor(my_distances)
```

```
import numpy as np
numpy_array = np.array([1.0, 2.0, 3.0])
tensor_from_numpy = torch.from_numpy(numpy_array)
```

```
zeros = torch.zeros(3, 3)     # 3x3 tensor of zeros
ones = torch.ones(2, 4)       # 2x4 tensor of ones
random = torch.rand(5, 5)     # 5x5 tensor with random values
```

Reshaping Tensors

```
C:>
torch.Size([6, 1])
```

Reshaping Tensors

```
# Your delivery data
single_distance = torch.tensor(25.0)
print(single_distance.shape)
simple_model = nn.Linear(1, 1)
```

```
C:>
torch.Size([])
```

No dimensions!

Model expects: torch.size(**[batch_size, 1]**)

Reshaping Tensors

```
# Your delivery data
single_distance = torch.tensor(25.0)

with_batch = single_distance.unsqueeze(0)
# torch.Size([1]) — added batch dim

ready_for_model = with_batch.unsqueeze(1)
# torch.Size([1, 1]) — now ready!
```

```
tensor([[25.0]])
```

Reshaping Tensors

```
# Your delivery data
single_distance = torch.tensor(25.0)

with_batch = single_distance.unsqueeze(0)
# torch.Size([1]) – added batch dim

ready_for_model = with_batch.unsqueeze(1)
# torch.Size([1, 1]) – now ready!
```

```
# Your delivery data
squeezed = ready_for_model.squeeze()
```

```
tensor(25.0)
```

```
tensor([[25.0]])
```

```
print(my_tensor.shape)
```

Indexing and Slicing

```
# Your predictions from the delivery model
predictions = torch.tensor([[14.9], [24.1], [35.6], [45.2]])
```

Indexing and Slicing

```
# Your predictions from the delivery model
predictions = torch.tensor([[14.9], [24.1], [35.6], [45.2]])

# Get the first prediction
first = predictions[0]
print(first)
```

Output:

```
C:>
tensor([14.9])
```

Indexing and Slicing

```
# Your predictions from the delivery model
predictions = torch.tensor([[14.9], [24.1], [35.6], [45.2]])

# Get the first prediction
first = predictions[0]
print(first)

# Get first three predictions
first_three = predictions[:3]
print(first_three)
```

Output:

```
C:>
tensor([[14.9000],
        [24.1000],
        [35.6000]])
```

Indexing and Slicing

```
# Your predictions from the delivery model
predictions = torch.tensor([[14.9], [24.1], [35.6], [45.2]])

# Get the first prediction
first = predictions[0]
print(first)

# Get first three predictions
first_three = predictions[:3]
print(first_three)

# Use .item() to get a Python number
value = predictions[0].item()
```

Indexing and Slicing

```
# Your predictions from the delivery model
predictions = torch.tensor([[14.9], [24.1], [35.6], [45.2]])

# Get the first prediction
first = predictions[0]
print(first)

# Get first three predictions
first_three = predictions[:3]
print(first_three)

# Use .item() to get a Python number
value = predictions[0].item()
```

```
C:>
14.899999618530273
<class 'float'>
```

Indexing and Slicing

```
# Multiple features per sample
data = torch.tensor([[3.0, 8.0, 1.0],     # distance, hour, weather
                     [7.0, 17.0, 2.0],
                     [12.0, 12.0, 1.0]])
```

```
distances = data[:, 0]
```

Output:

```
C:>
tensor([ 3.,  7., 12.])
```

Tensors

- Shapes
- Types
- Reshaping
- Indexing

DeepLearning.AI

Tensor Math and Broadcasting

Getting Started with PyTorch

Tensor Math

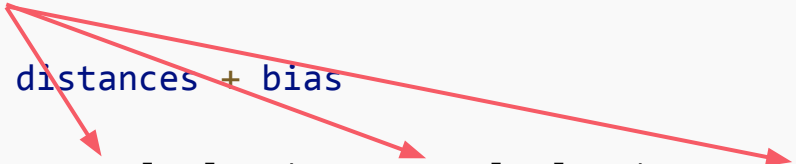
```
distances = [[3.0], [7.0], [12.0]]
weight = 2.3
bias = 8.0

weight * distances + bias

[[3.0], [7.0], [12.0]]
```

Tensor Math

```
distances = [[3.0], [7.0], [12.0]]
weight = 2.3
bias = 8.0

weight * distances + bias

[[2.3*3.0], [2.3*7.0], [2.3*12.0]]
```

Tensor Math

```
distances = [[3.0], [7.0], [12.0]]
weight = 2.3
bias = 8.0

weight * distances + bias

[[2.3*3.0 + 8.0], [2.3*7.0 + 8.0], [2.3*12.0 + 8.0]]
```



Tensor Math

Scalars

```
2.3 * [[3.0], [7.0], [12.0]] = [[2.3*3.0], [2.3*7.0], [2.3*12.0]]
```

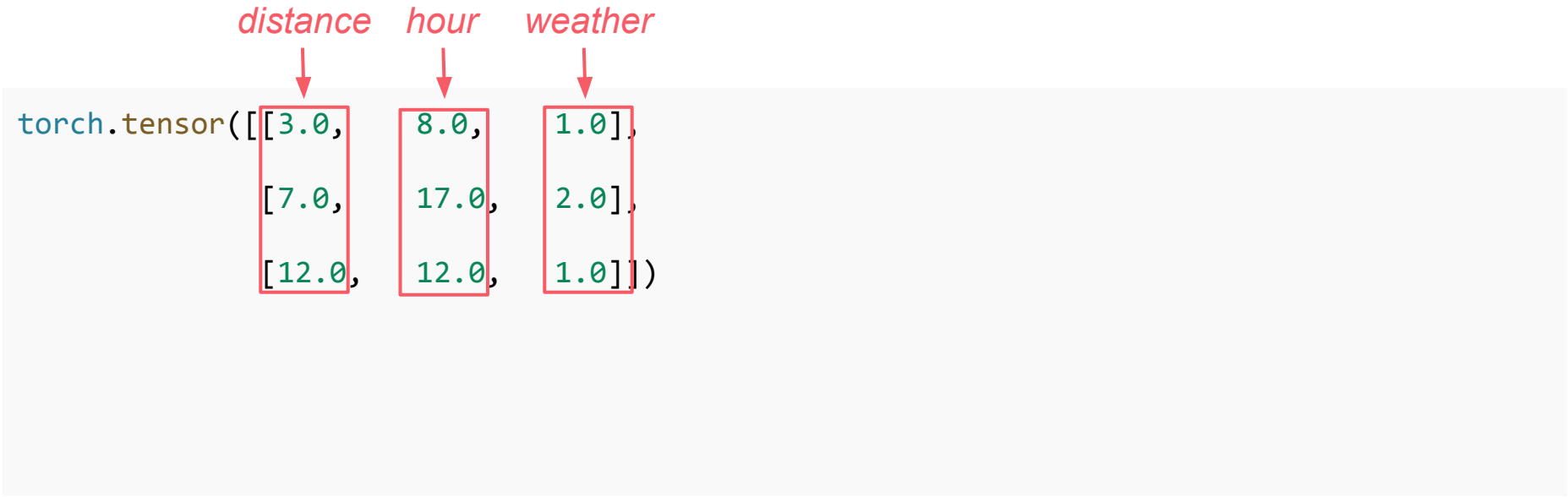
Tensors with same Shape

```
[[1.0], [2.0], [3.0]] * [[3.0], [7.0], [12.0]]  = [[1.0*3.0], [2.0*7.0], [3.0*12.0]]
```

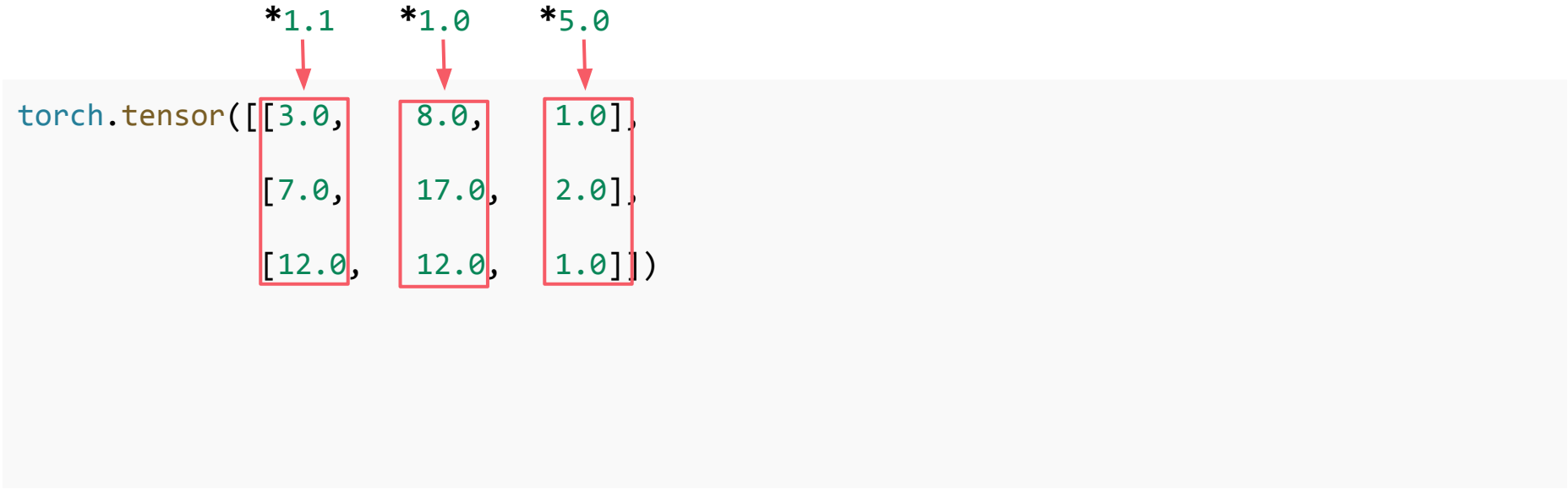
Tensor Math

```
torch.tensor([[3.0,    8.0,    1.0],   <-- 1
              [7.0,    17.0,   2.0],   <-- 2
              [12.0,   12.0,   1.0]])  <-- 3
```

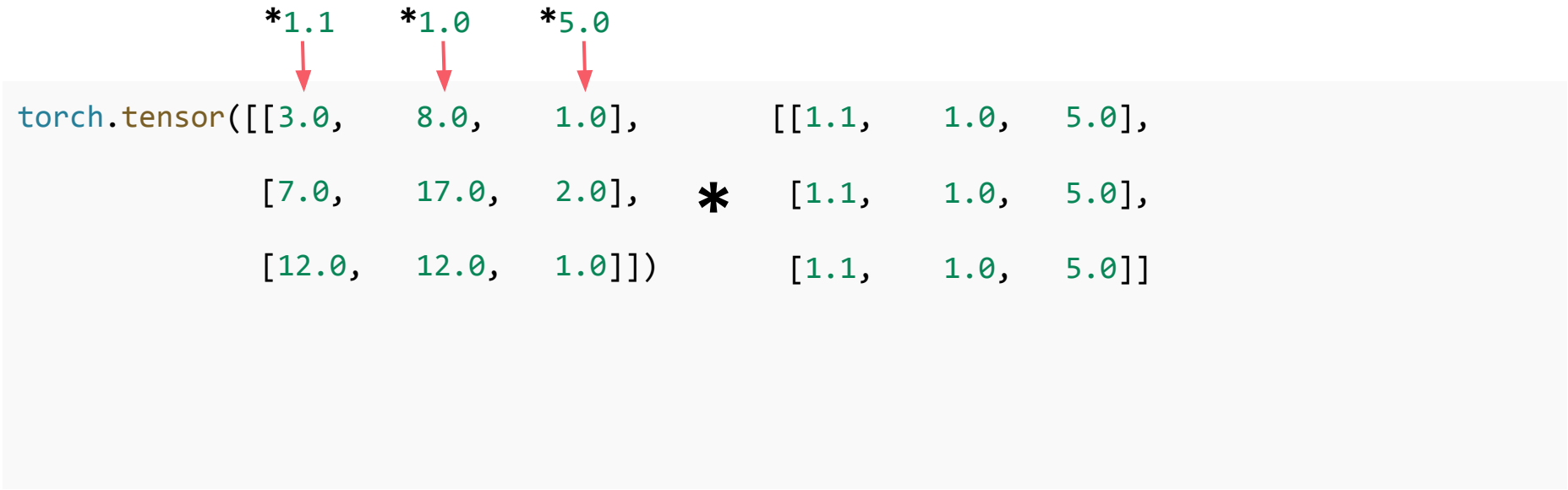
Tensor Math


distance
hour
weather
torch.tensor([[3.0, 8.0, 1.0],
[7.0, 17.0, 2.0],
[12.0, 12.0, 1.0]])

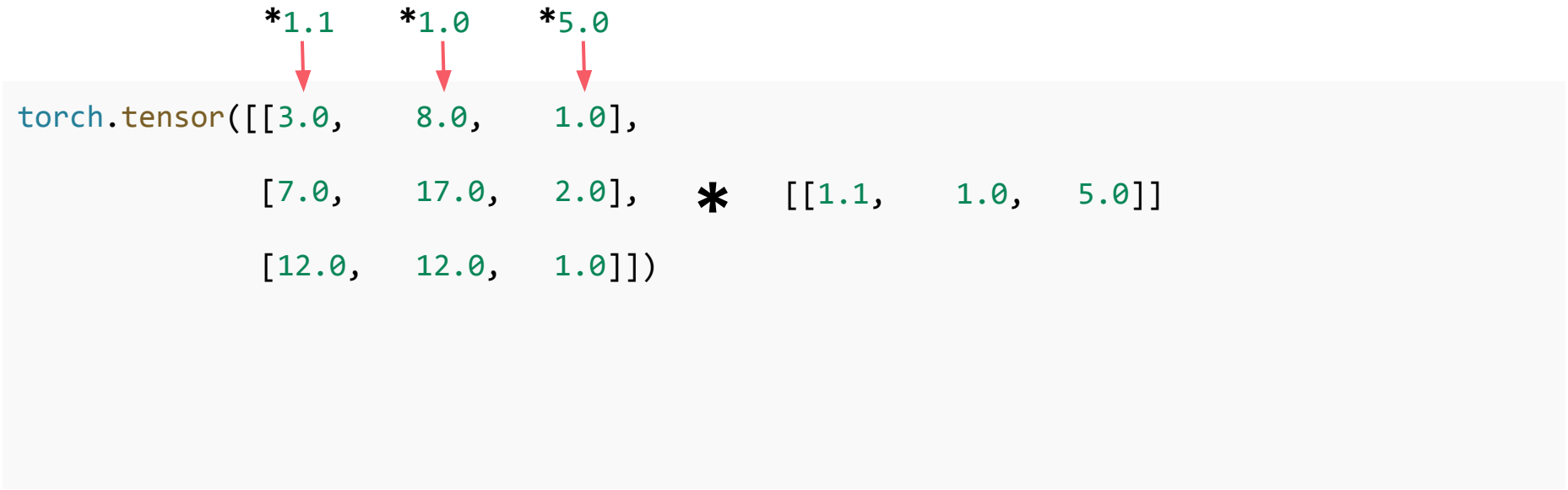
Tensor Math



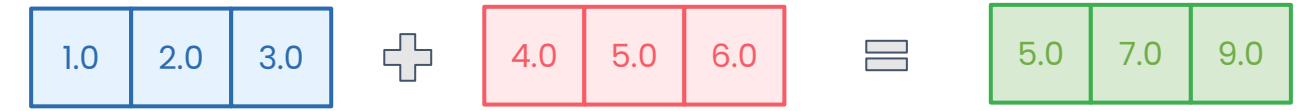
Tensor Math


*1.1
*1.0
*5.0
torch.tensor([[3.0, 8.0, 1.0],
[7.0, 17.0, 2.0],
[12.0, 12.0, 1.0]])
*
[[1.1, 1.0, 5.0],
[1.1, 1.0, 5.0],
[1.1, 1.0, 5.0]]

Tensor Math



Broadcasting



`[[1.0, 2.0, 3.0]]`

shape = (1 , 3)

`[[4.0, 5.0, 6.0]]`

shape = (1 , 3)

`[[5.0, 7.0, 9.0]]`

shape = (1 , 3)

Broadcasting

`[[1.0, 2.0, 3.0]]`

shape = (1 , 3)

`5.0`

shape = ( )

`[[6.0, 7.0, 8.0]]`

shape = (1 , 3)

Broadcasting

`[[1.0, 2.0, 3.0]]`

shape = (1 , 3)

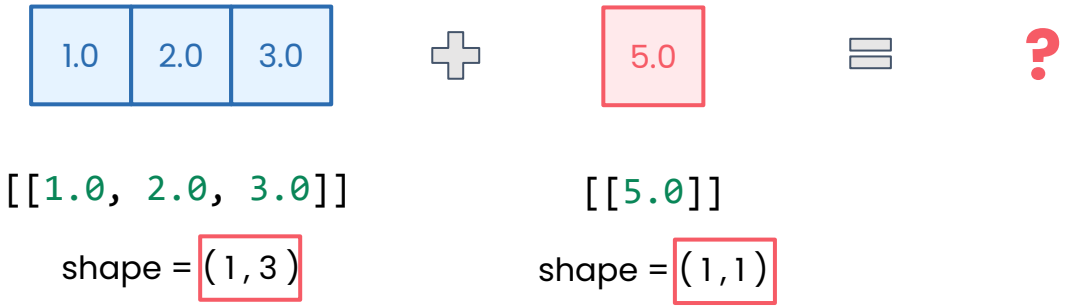
`[[5.0, 5.0, 5.0]]`

shape = (1 , 3)

`[[6.0, 7.0, 8.0]]`

shape = (1 , 3)

Broadcasting



Broadcasting


```
[[1.0, 2.0, 3.0]]
```

shape = (1, 3)

```
[[5.0]]
```

shape = (1, 1)

Broadcasting

`[[1.0, 2.0, 3.0]]`

shape = (1 , 3)

`[[5.0]]`

shape = (1 , 3)

Broadcasting

`[[1.0, 2.0, 3.0]]`

shape = (1 , 3)

`[[5.0, 5.0, 5.0]]`

shape = (1 , 3)

Broadcasting

`[[1.0, 2.0, 3.0]]`

shape = (1 , 3)

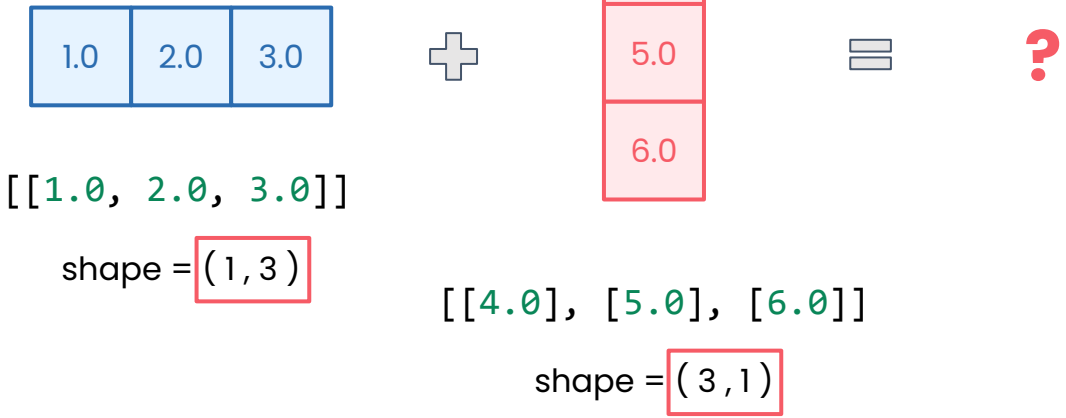
`[[5.0, 5.0, 5.0]]`

shape = (1 , 3)

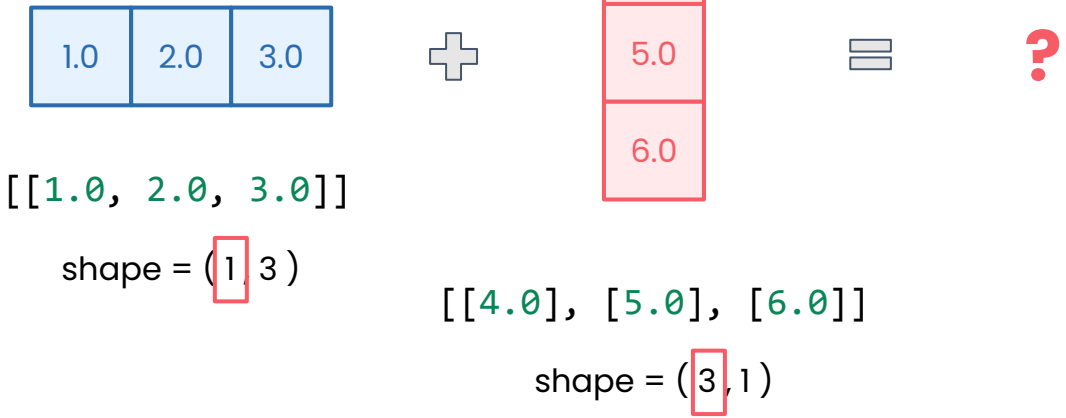
`[[6.0, 7.0, 8.0]]`

shape = (1 , 3)

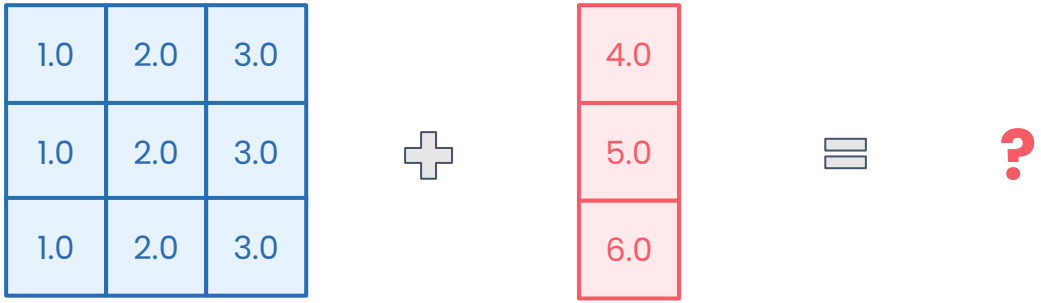
Broadcasting



Broadcasting



Broadcasting



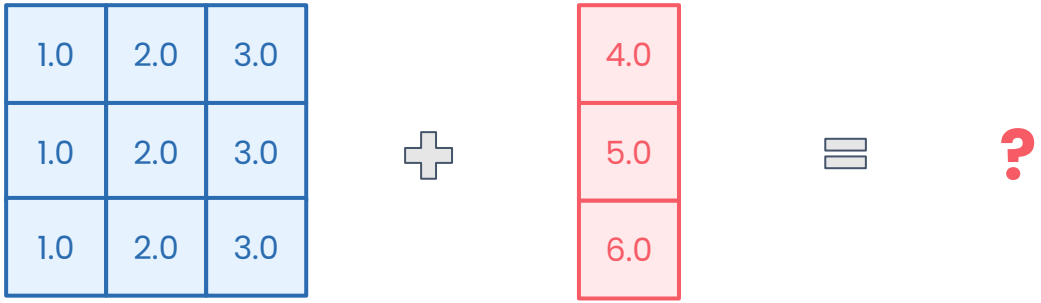

```
[[1.0, 2.0, 3.0],
 [1.0, 2.0, 3.0],
 [1.0, 2.0, 3.0]]
```

shape = (3 , 3)

```
[[4.0], [5.0], [6.0]]
```

shape = (3 , 1)

Broadcasting



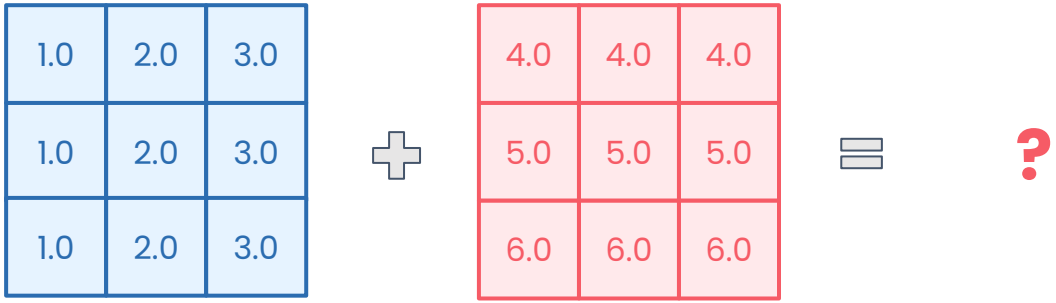

```
[[1.0, 2.0, 3.0],
 [1.0, 2.0, 3.0],
 [1.0, 2.0, 3.0]]
```

shape = (3 , 3)

```
[[4.0], [5.0], [6.0]]
```

shape = (3 , 1)

Broadcasting



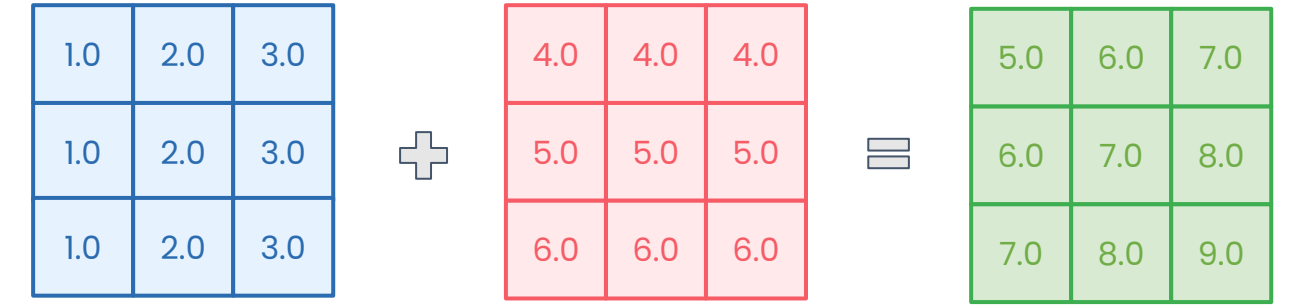

```
[[1.0, 2.0, 3.0],
 [1.0, 2.0, 3.0],
 [1.0, 2.0, 3.0]]
```

shape = (3 , 3)

```
[[4.0, 4.0, 4.0],
 [5.0, 5.0, 5.0],
 [6.0, 6.0, 6.0]]
```

shape = (3 , 3)

Broadcasting




```
[[1.0, 2.0, 3.0],
 [1.0, 2.0, 3.0],
 [1.0, 2.0, 3.0]]
```

shape = (3 , 3)

```
[[4.0, 4.0, 4.0],
 [5.0, 5.0, 5.0],
 [6.0, 6.0, 6.0]]
```

shape = (3 , 3)

```
[[5.0, 6.0, 7.0],
 [6.0, 7.0, 8.0],
 [7.0, 8.0, 9.0]]
```

shape = (3 , 3)

Tensor Math

```
torch.tensor([[3.0,    8.0,    1.0],        [[1.1,    1.0,    5.0],
              [7.0,    17.0,   2.0],    *    [1.1,    1.0,    5.0],
              [12.0,   12.0,   1.0]])       [1.1,    1.0,    5.0]]
```

Tensor Math

```
torch.tensor([[3.0,    8.0,    1.0],
              [7.0,    17.0,   2.0],   *   [[1.1,    1.0,   5.0]]
              [12.0,   12.0,   1.0]])
```