

# Introduction to deep learning with PyTorch

INTRODUCTION TO DEEP LEARNING WITH PYTORCH

Jasmin Ludolf

Senior Data Science Content Developer,  
DataCamp



# Deep learning is everywhere!



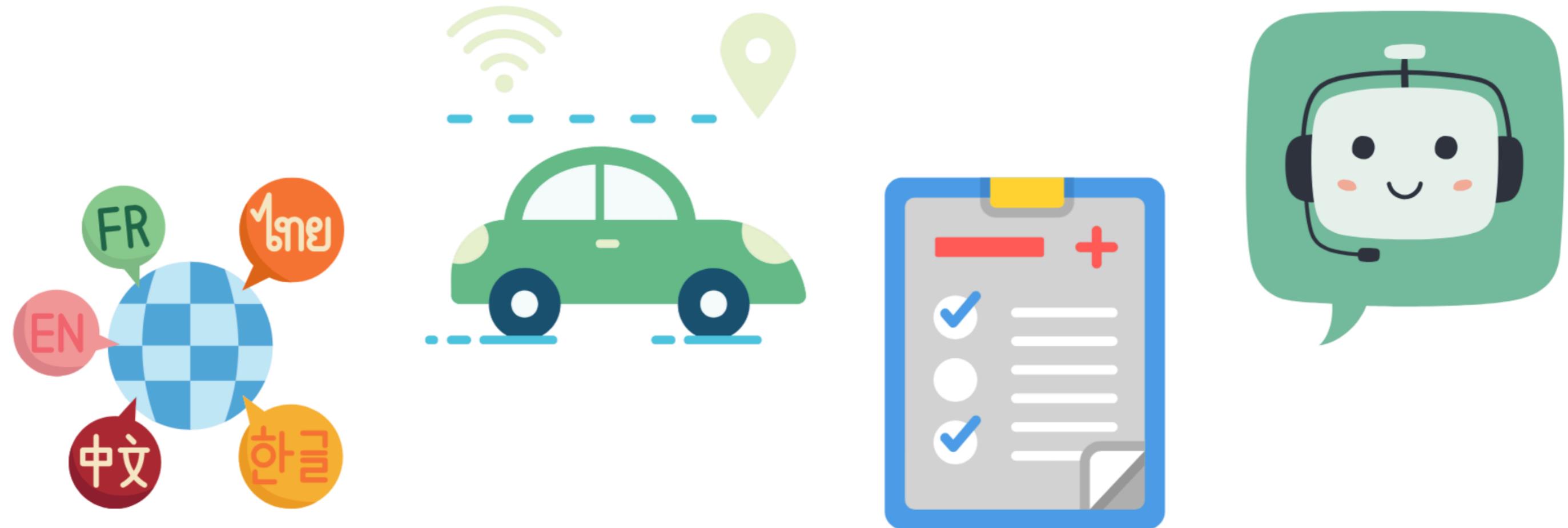
# Deep learning is everywhere!



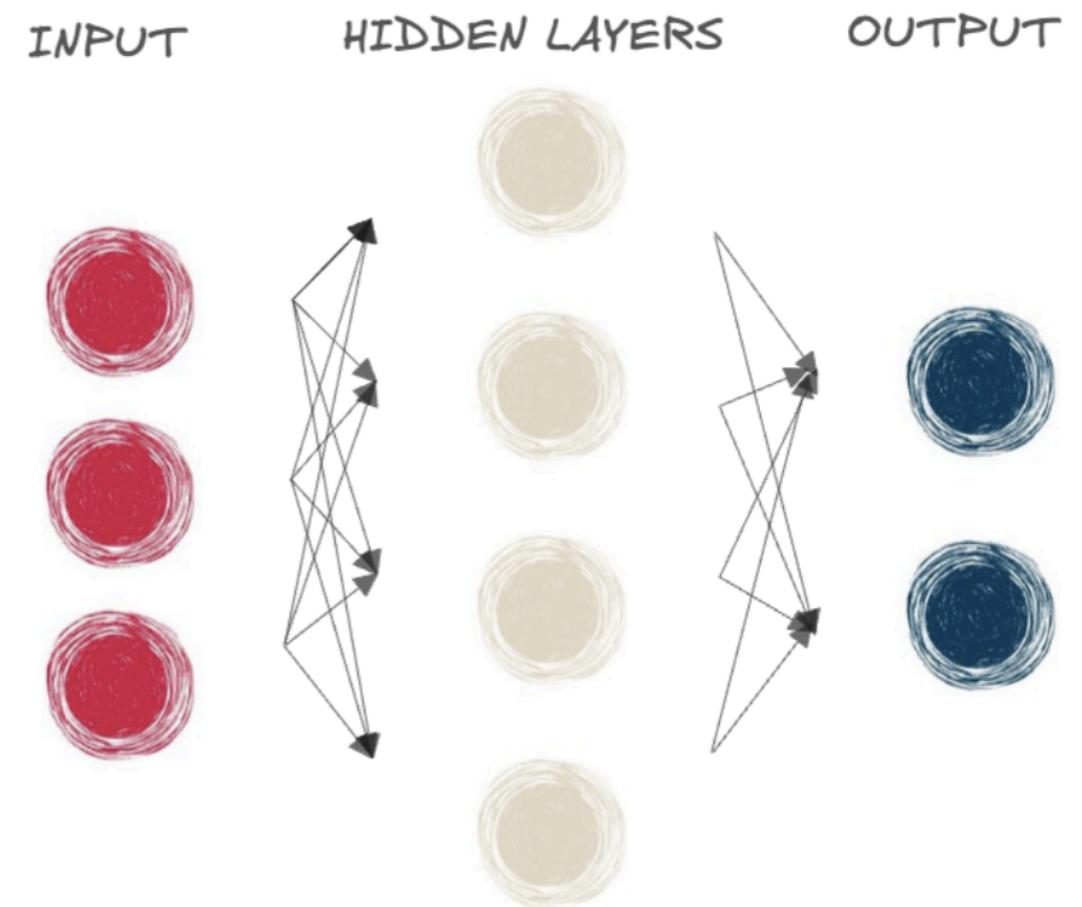
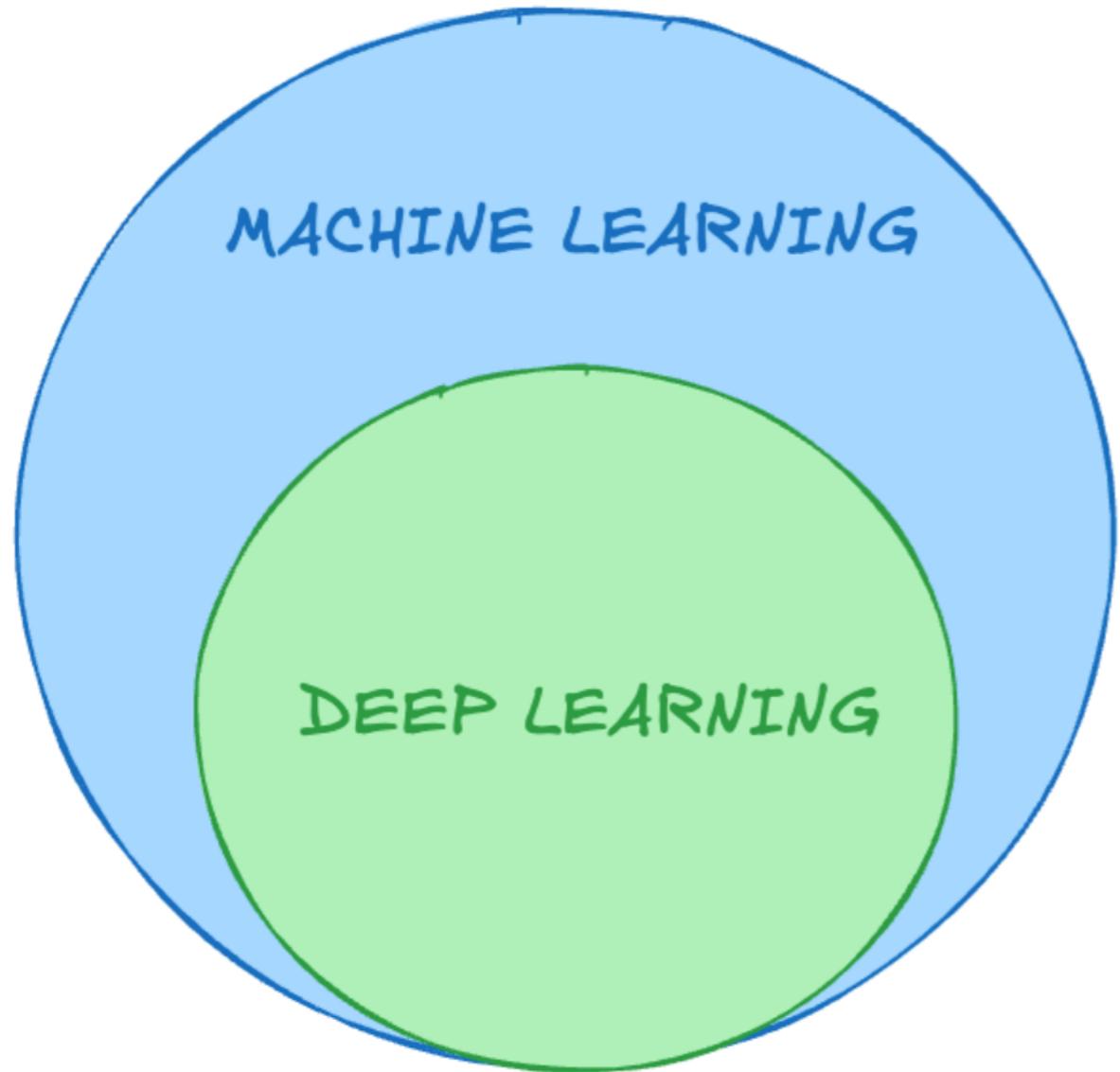
# Deep learning is everywhere!



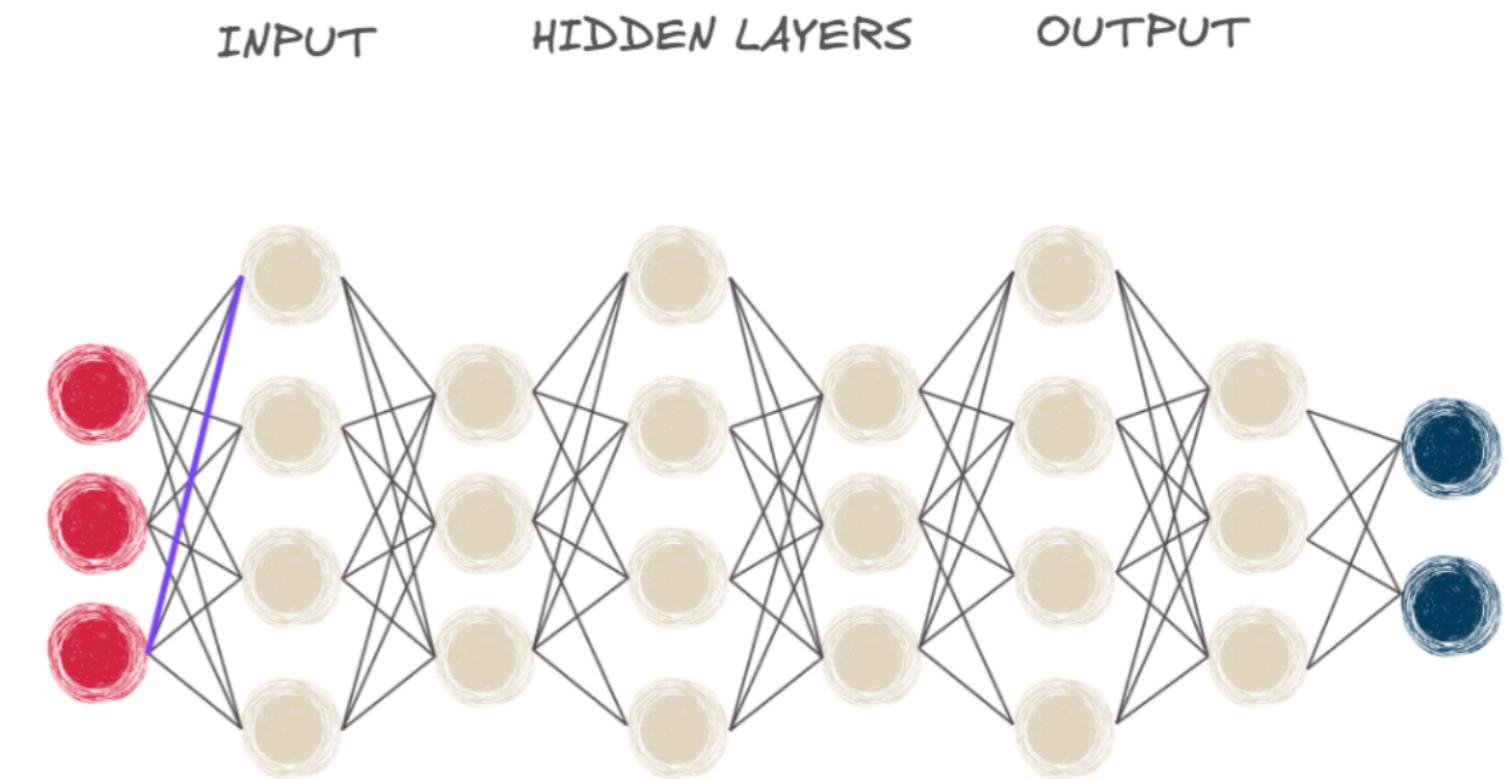
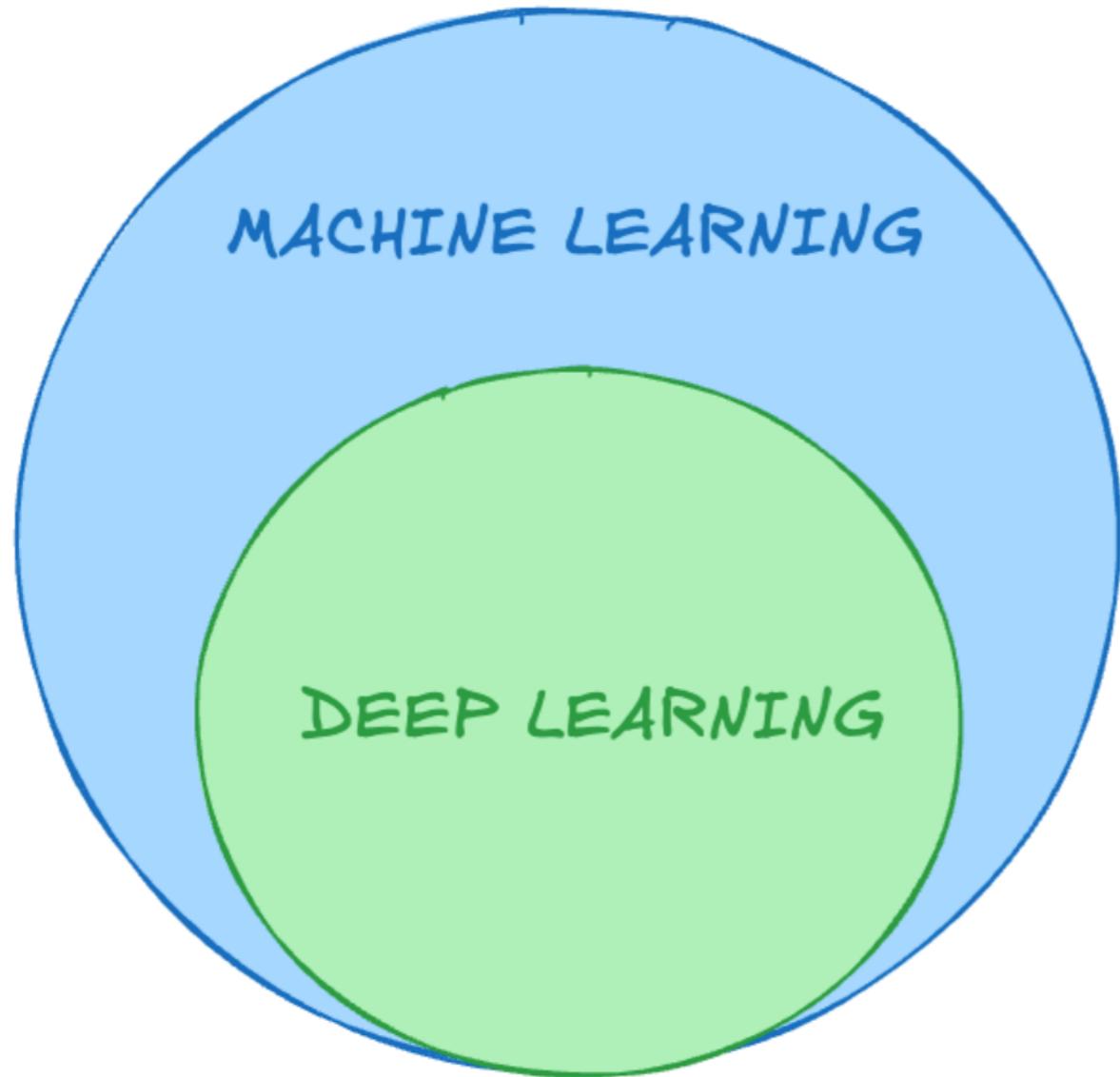
# Deep learning is everywhere!



# What is deep learning?



# What is deep learning?



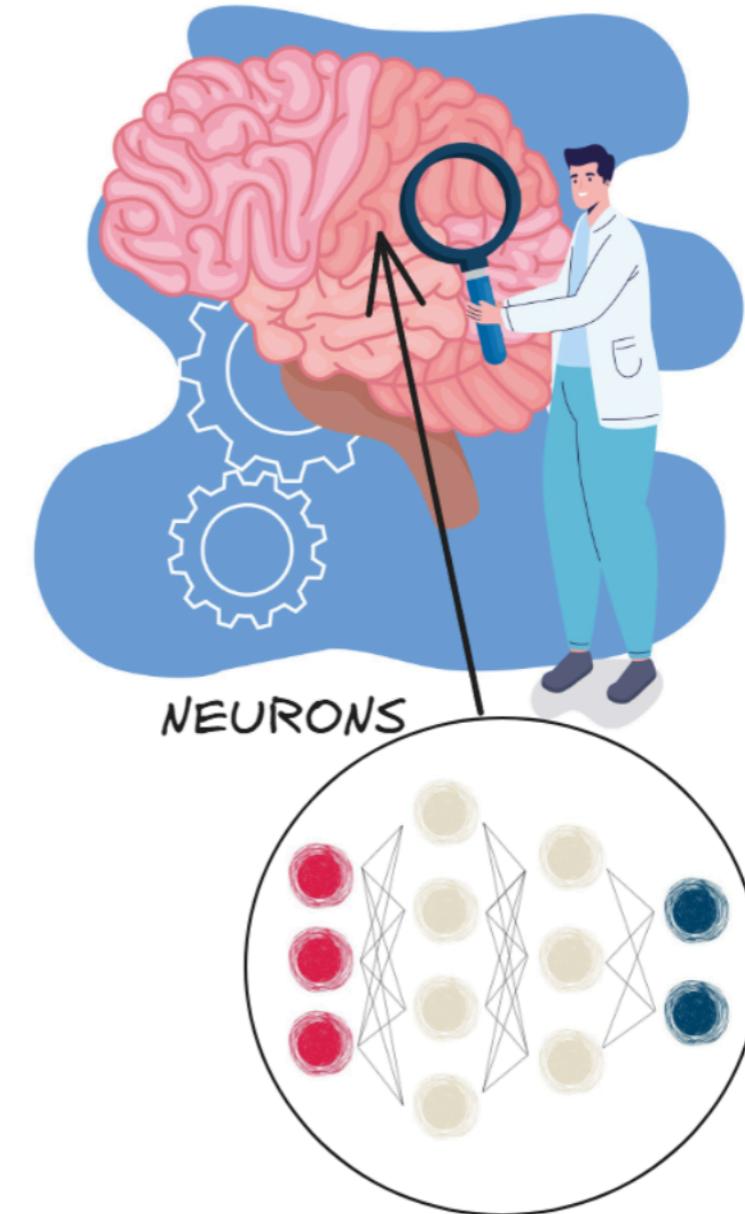
# Deep learning networks

- Inspired by how the human brain learns



# Deep learning networks

- Inspired by how the human brain learns
- Neurons → neural networks
- Models require large amount of data
- At least 100,000s data points



# PyTorch: a deep learning framework

- One of the most popular frameworks
- Originally developed by Meta AI, now part of Linux Foundation
- Intuitive and user-friendly
- Similarities with NumPy



# PyTorch tensors

- Tensor:
  - Similar to array or matrix
  - Building block of neural networks

```
import torch  
my_list = [[1, 2, 3], [4, 5, 6]]  
tensor = torch.tensor(my_list)  
print(tensor)
```

```
tensor([[1, 2, 3],  
        [4, 5, 6]])
```

# Tensor attributes

- Tensor shape

```
my_list = [[1, 2, 3], [4, 5, 6]]  
tensor = torch.tensor(my_list)  
print(tensor.shape)
```

```
torch.Size([2, 3])
```

- Tensor data type

```
print(tensor.dtype)
```

```
torch.int64
```

# Getting started with tensor operations

## Compatible shapes

```
a = torch.tensor([[1, 1],  
                 [2, 2]])
```

```
b = torch.tensor([[2, 2],  
                 [3, 3]])
```

- Addition / subtraction

```
print(a + b)
```

```
tensor([[3, 3],  
       [5, 5]])
```

## Incompatible shapes

```
a = torch.tensor([[1, 1],  
                 [2, 2]])
```

```
c = torch.tensor([[2, 2, 4],  
                 [3, 3, 5]])
```

- Addition / subtraction

```
print(a + c)
```

```
RuntimeError: The size of tensor a  
(2) must match the size of tensor b (3)  
at non-singleton dimension 1
```

# Element-wise multiplication

```
a = torch.tensor([[1, 1],  
                  [2, 2]])  
  
b = torch.tensor([[2, 2],  
                  [3, 3]])  
  
print(a * b)
```

```
tensor([[2, 2],  
       [6, 6]])
```

# Matrix multiplication

```
a = torch.tensor([[1, 1],  
                  [2, 2]])  
  
b = torch.tensor([[2, 2],  
                  [3, 3]])  
  
print(a @ b)
```

```
tensor([[5, 5],  
        [10, 10]])
```

# Matrix multiplication

```
a = torch.tensor([[1, 1],  
                  [2, 2]])  
  
b = torch.tensor([[2, 2],  
                  [3, 3]])  
  
print(a @ b)
```

```
tensor([[5, 5],  
        [10, 10]])
```

- $1 \cdot 2 + 1 \cdot 3 = 5$
- Perform addition and multiplication to process data and learn patterns

# **Let's practice!**

**INTRODUCTION TO DEEP LEARNING WITH PYTORCH**

# Neural networks and layers

INTRODUCTION TO DEEP LEARNING WITH PYTORCH

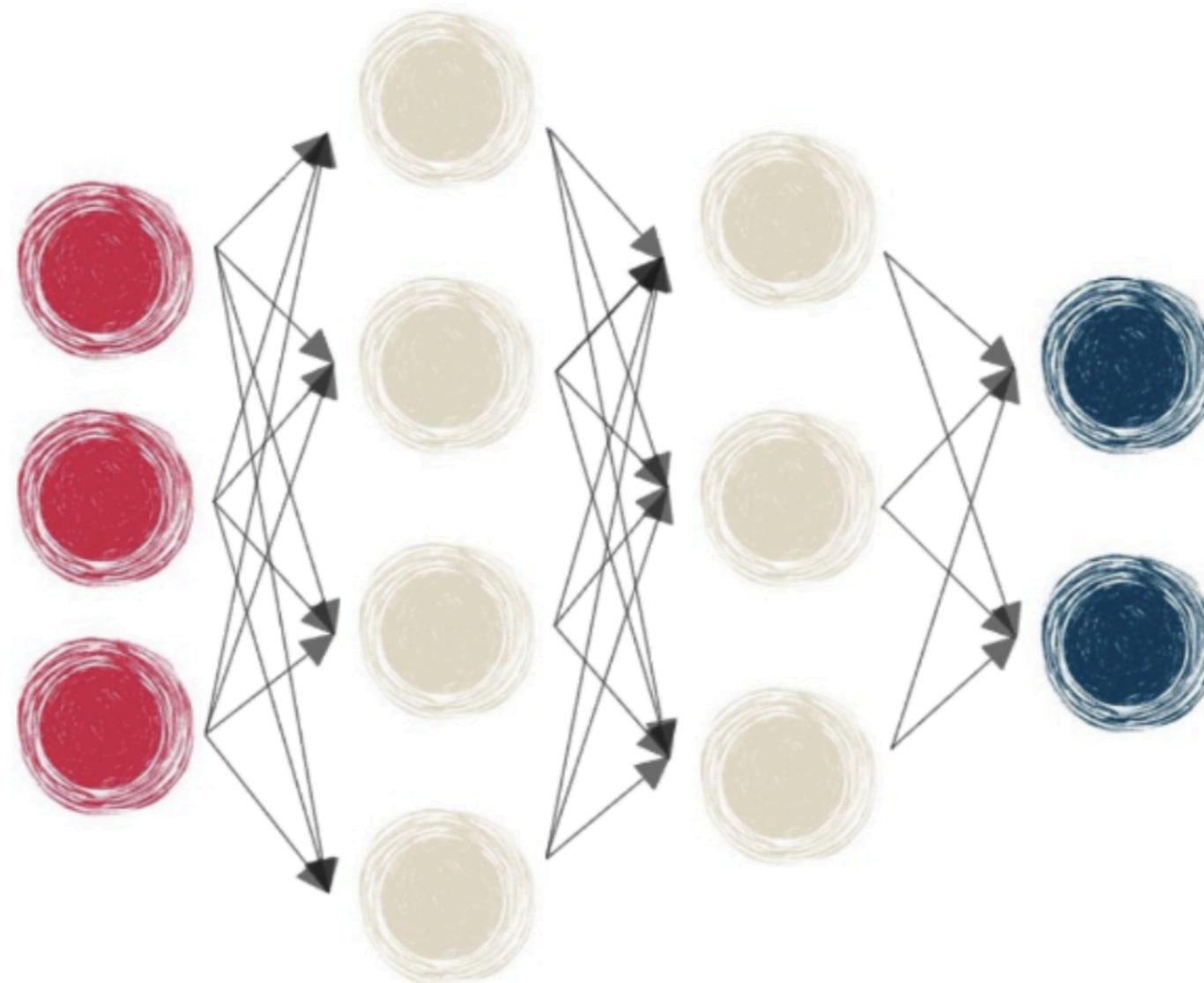


Jasmin Ludolf

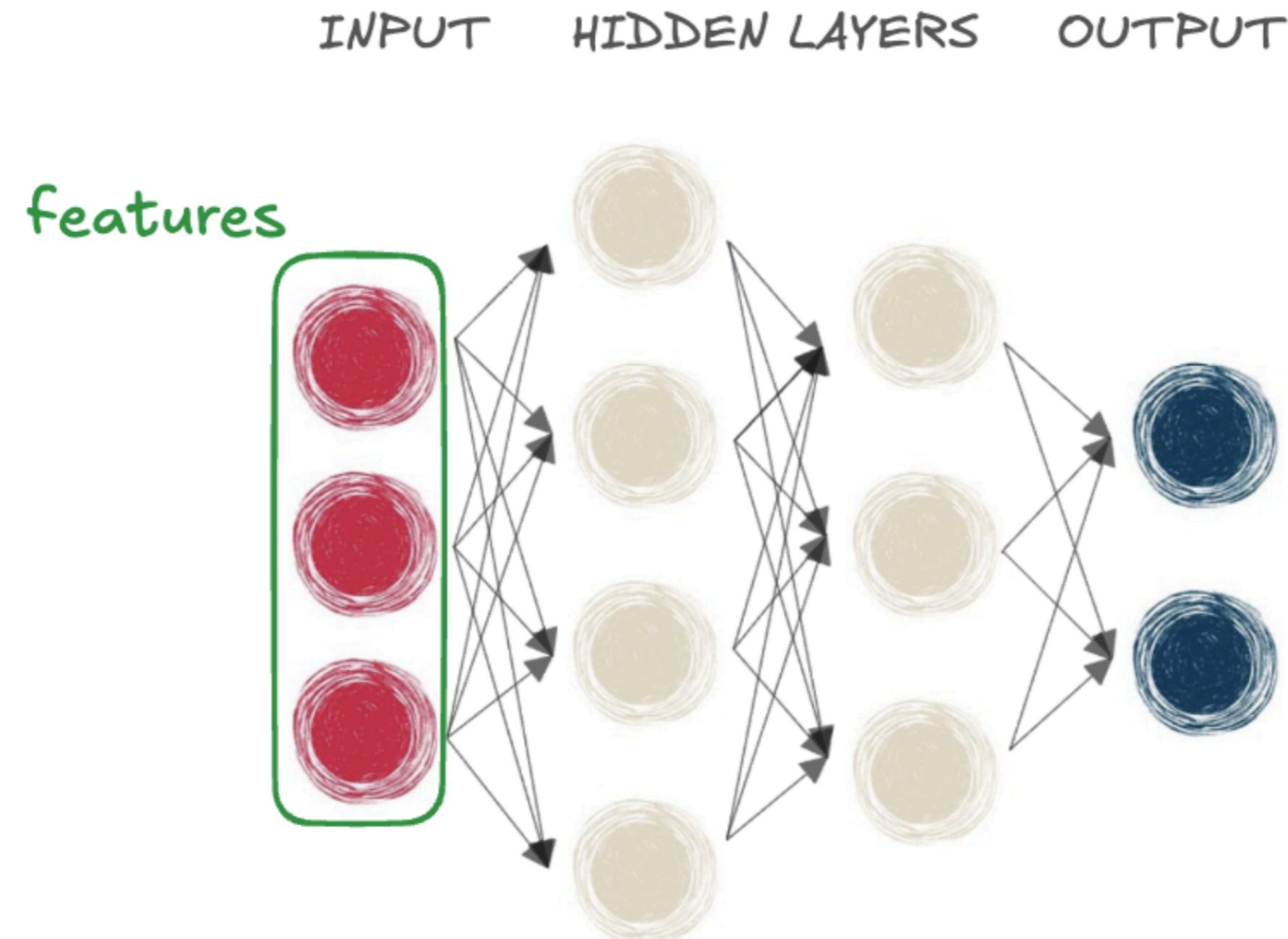
Senior Data Science Content Developer,  
DataCamp

# Neural network layers

INPUT      HIDDEN LAYERS      OUTPUT

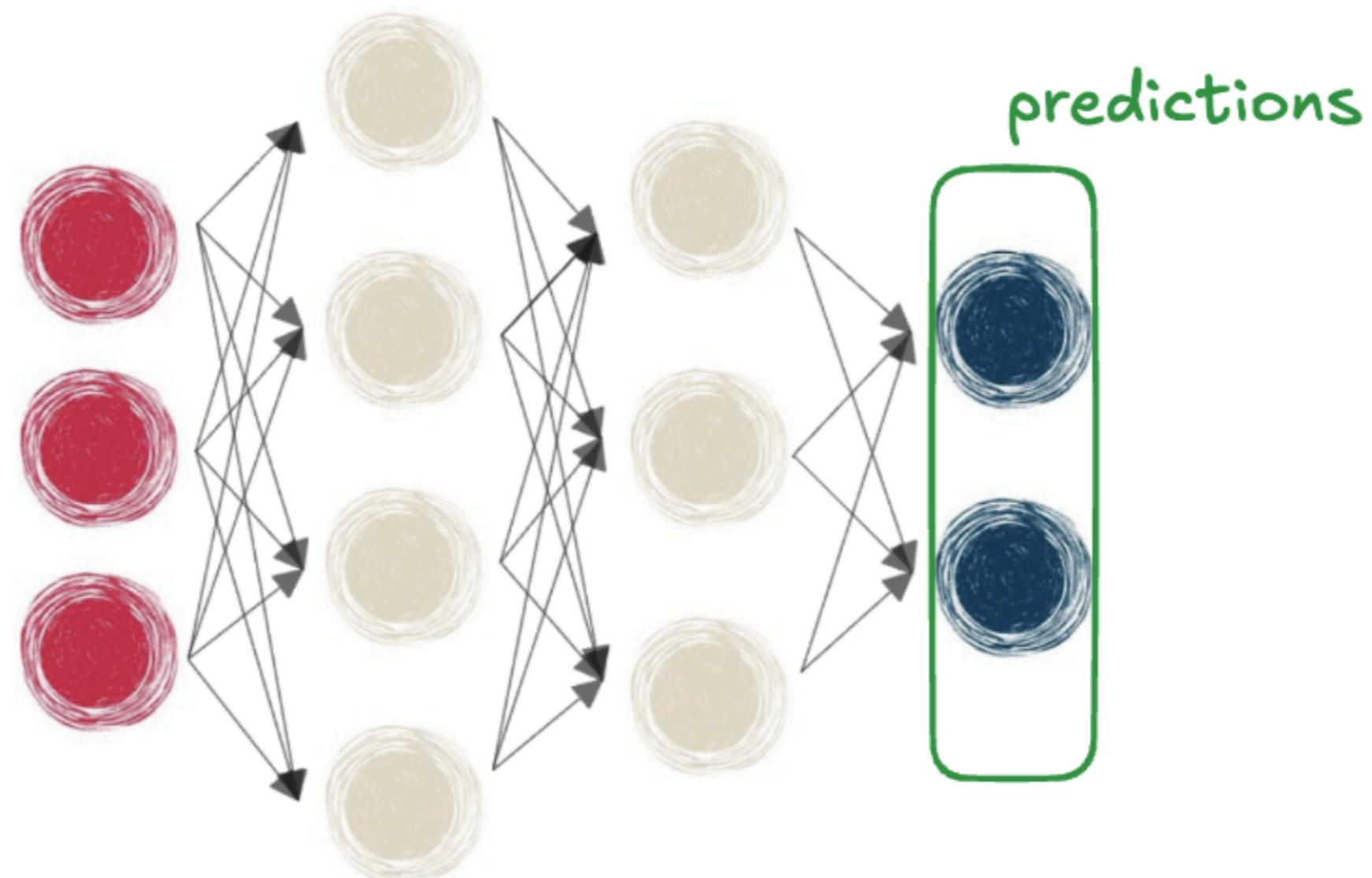


# Neural network layers



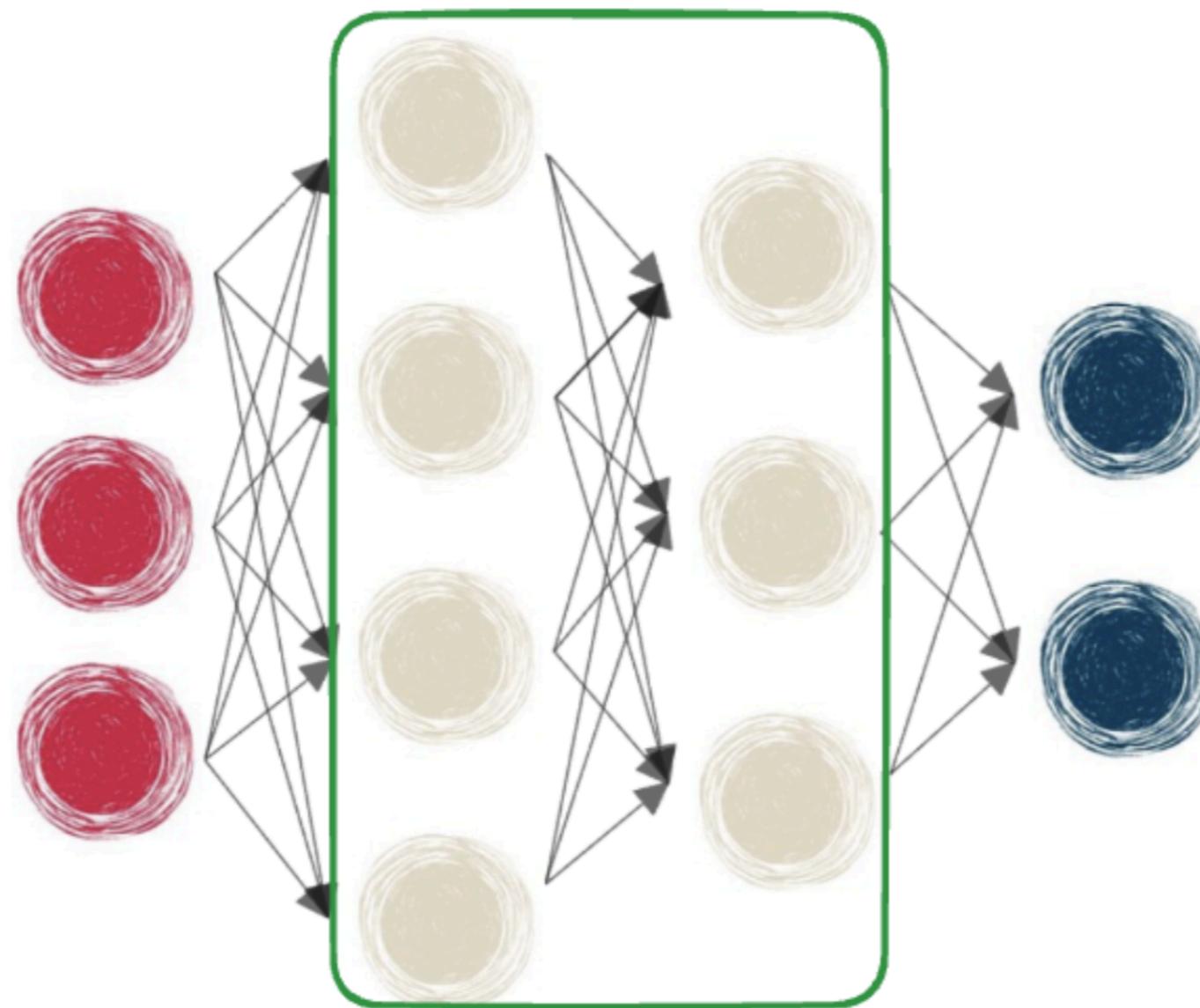
# Neural network layers

INPUT      HIDDEN LAYERS      OUTPUT

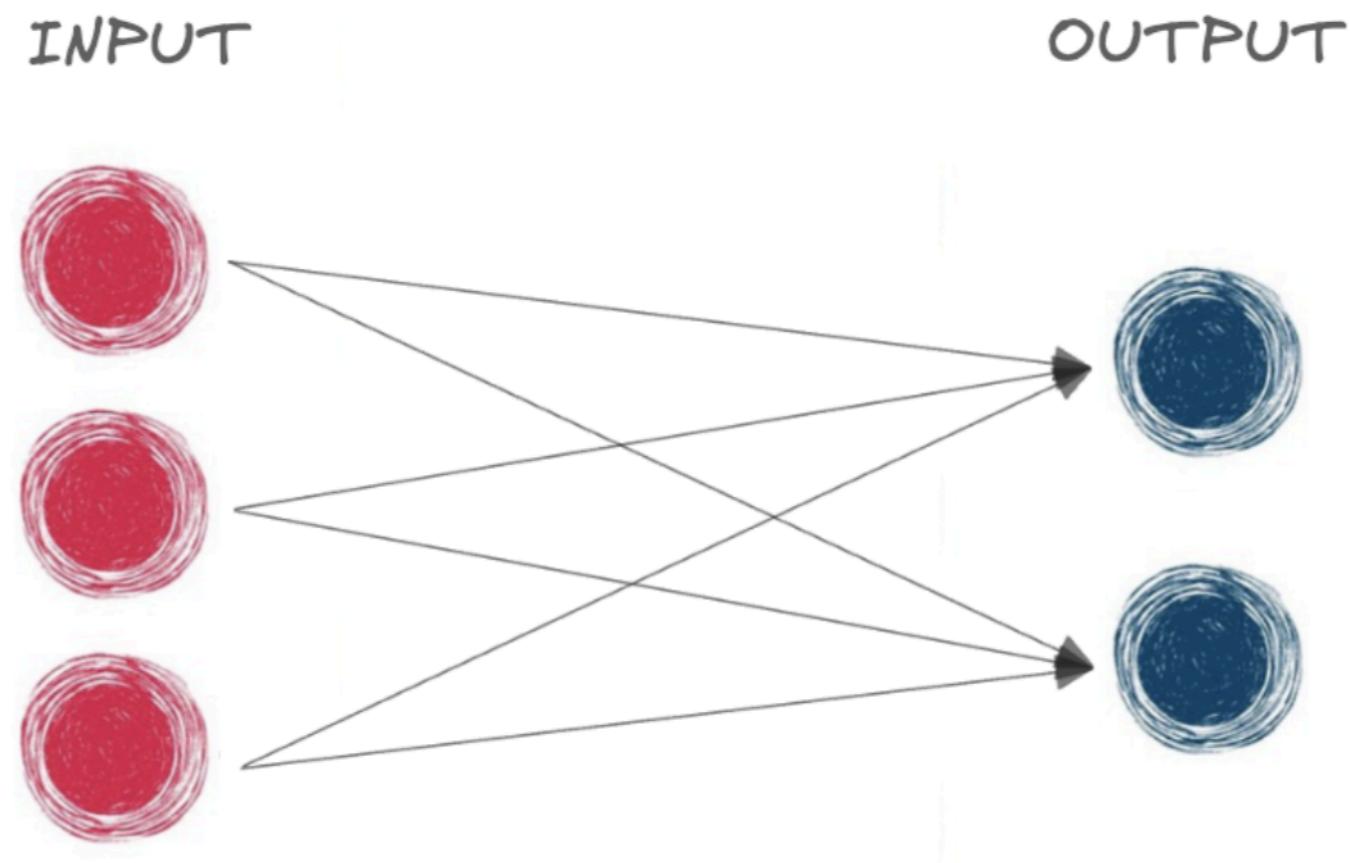


# Neural network layers

INPUT      HIDDEN LAYERS      OUTPUT



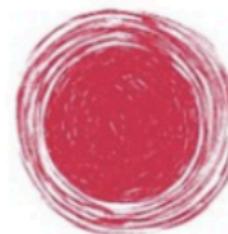
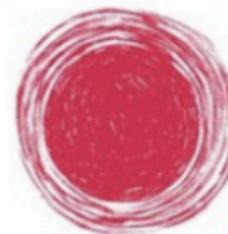
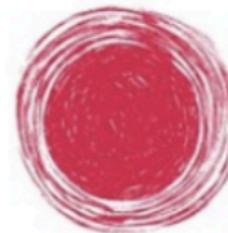
# Our first neural network



- Fully connected network
- Equivalent to a linear model

# Designing a neural network

INPUT

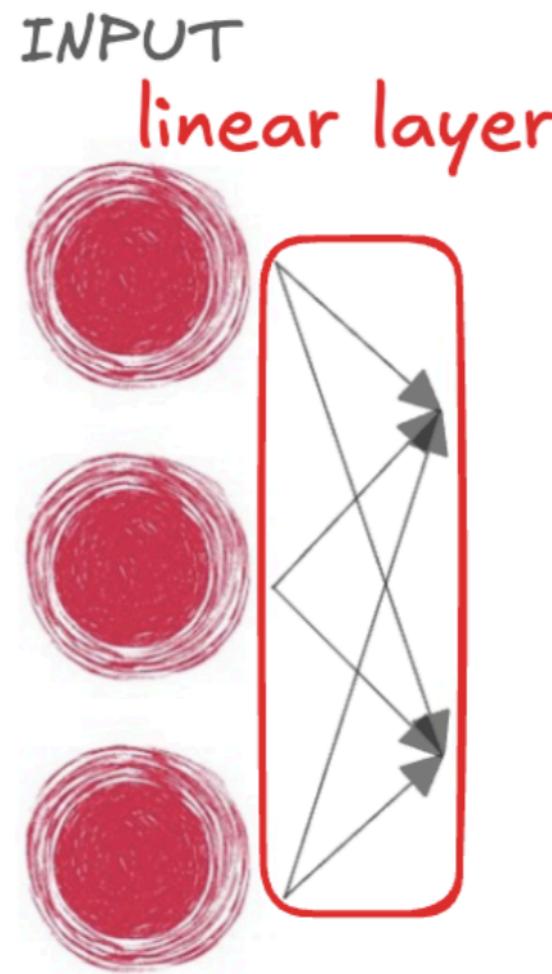


```
# Importing as nn to avoid writing torch.nn  
import torch.nn as nn
```

```
# Create input_tensor with three features  
input_tensor = torch.tensor(  
    [[0.3471, 0.4547, -0.2356]])
```

- Input neurons = features
- Output neurons = classes

# Designing a neural network



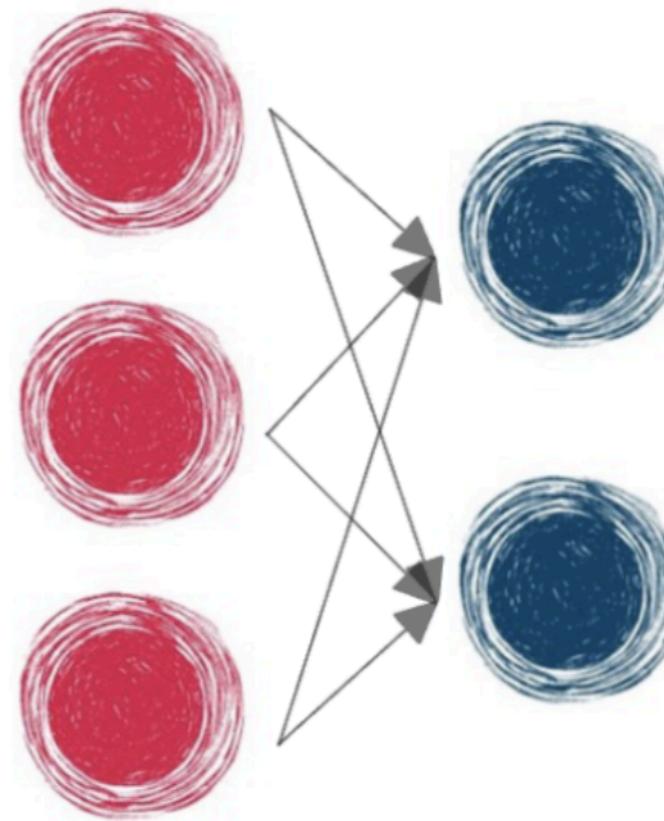
```
# Importing as nn to avoid writing torch.nn
import torch.nn as nn

# Create input_tensor with three features
input_tensor = torch.tensor(
    [[0.3471, 0.4547, -0.2356]])

# Define our linear layer
linear_layer = nn.Linear(
    in_features=3,
    out_features=2
)
```

# Designing a neural network

INPUT      OUTPUT



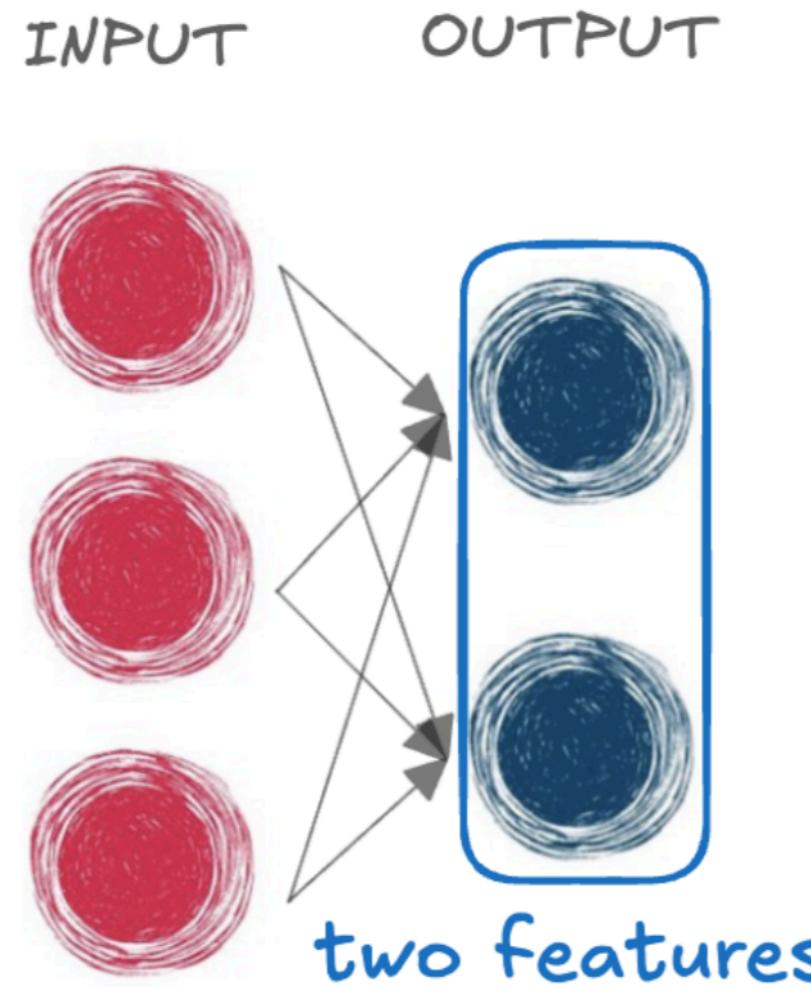
```
# Importing as nn to avoid writing torch.nn
import torch.nn as nn

# Create input_tensor with three features
input_tensor = torch.tensor(
    [[0.3471, 0.4547, -0.2356]])

# Define our linear layer
linear_layer = nn.Linear(
    in_features=3,
    out_features=2
)

# Pass input through linear layer
output = linear_layer(input_tensor)
print(output)
```

# Designing a neural network

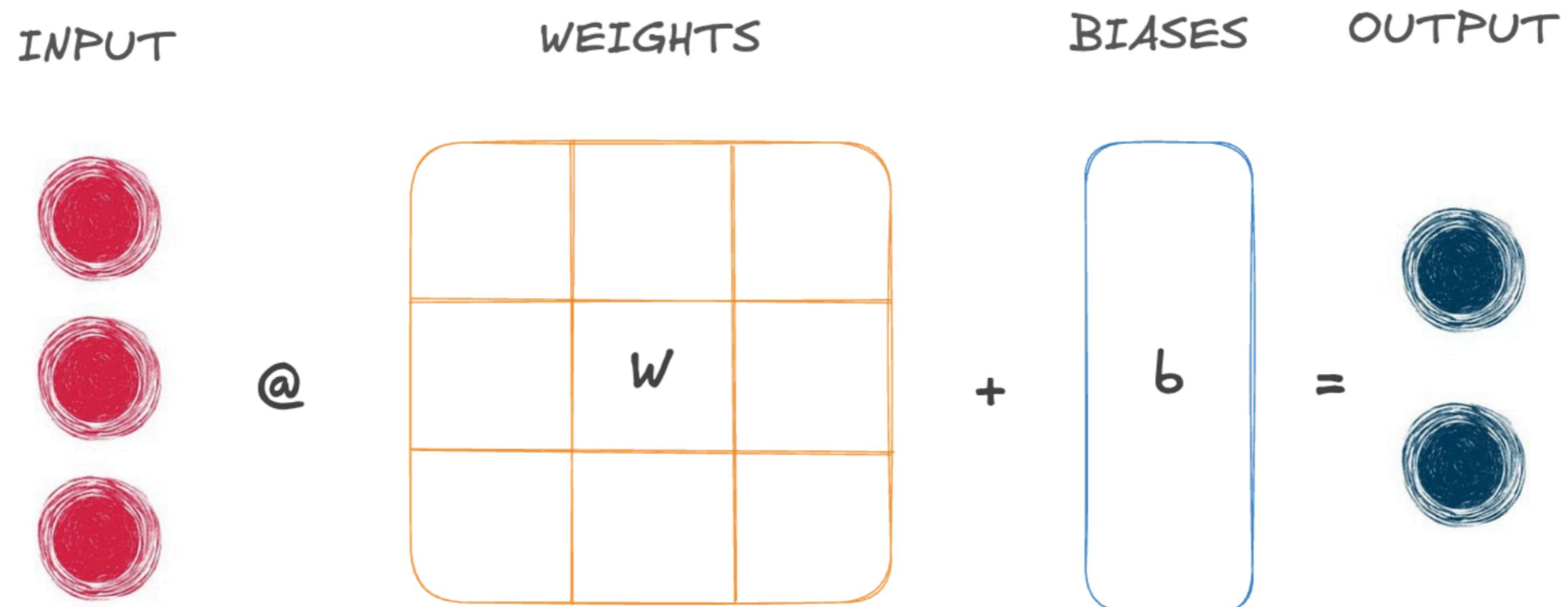


```
# Pass input through linear layer  
output = linear_layer(input_tensor)  
print(output)
```

```
tensor([[-0.2415, -0.1604]],  
grad_fn=<AddmmBackward0>)
```

# Weights and biases

```
output = linear_layer(input_tensor)
```



# Weights and biases

- `.weight`

```
print(linear_layer.weight)
```

Parameter containing:

```
tensor([[-0.4799,  0.4996,  0.1123],  
       [-0.0365, -0.1855,  0.0432]],  
      requires_grad=True)
```

- `.bias`

```
print(linear_layer.bias)
```

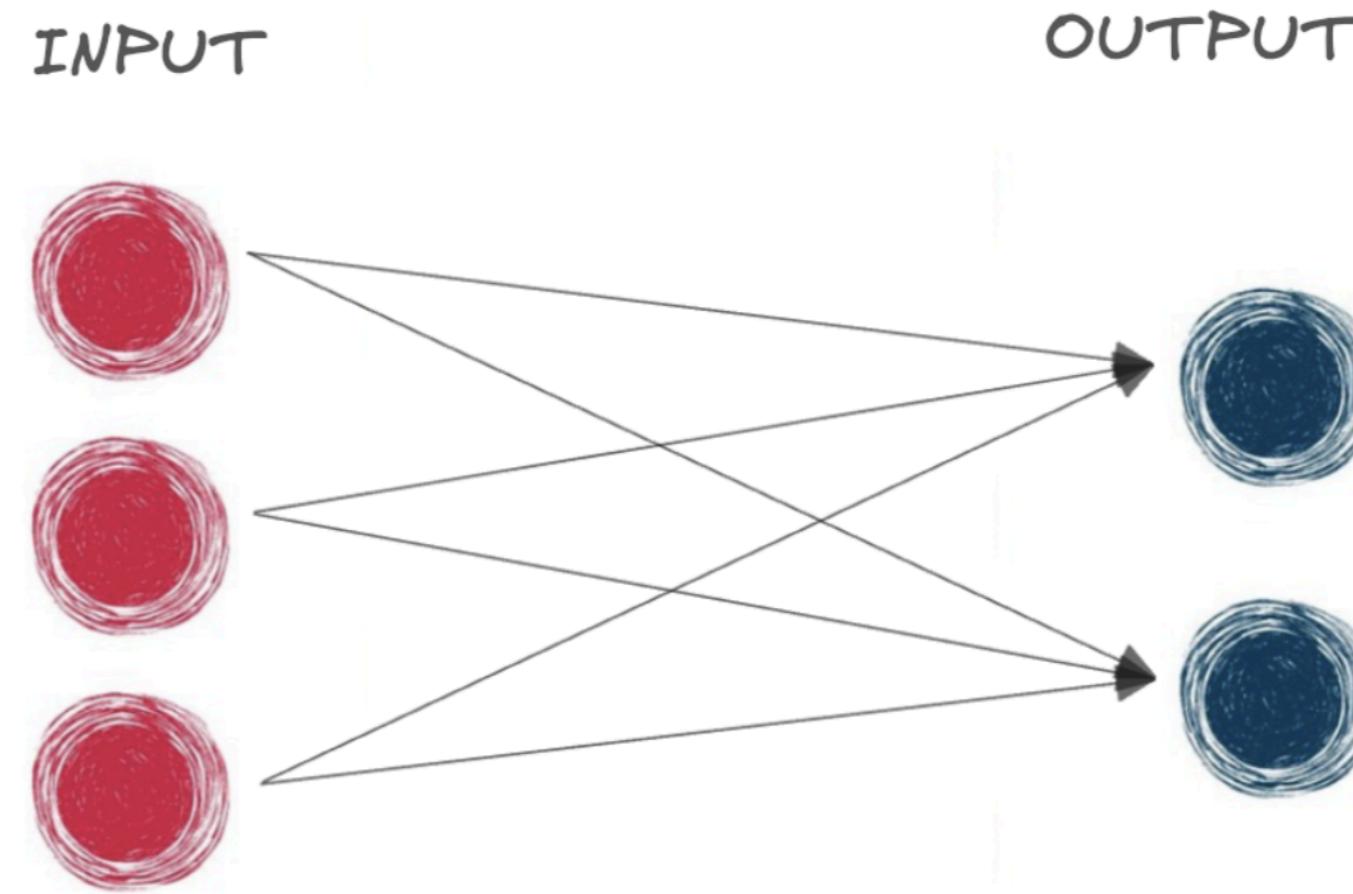
Parameter containing:

```
tensor([0.0310,  0.1537], requires_grad=True)
```

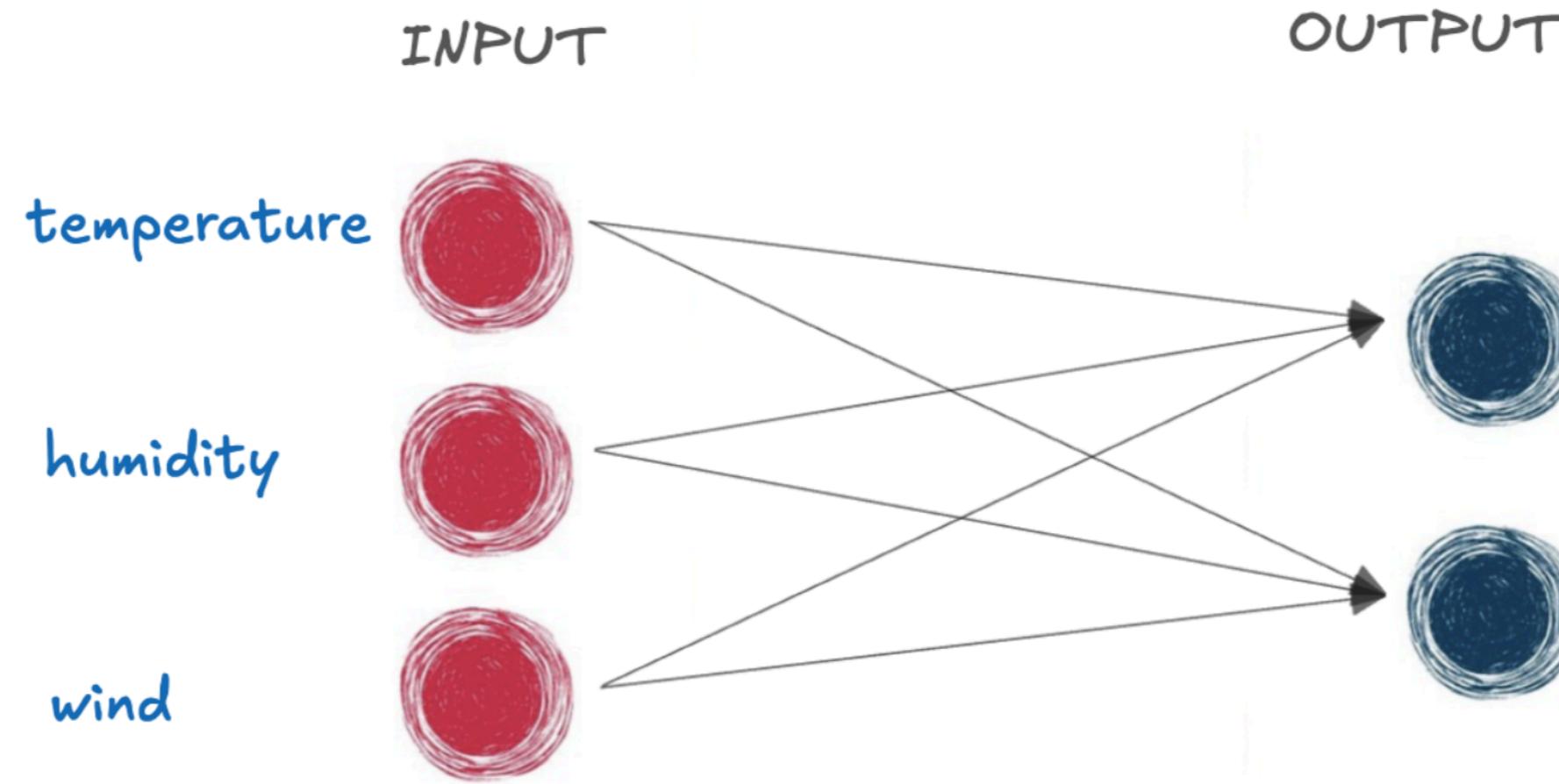
- Reflects the importance of different features

- Provides the neuron with a baseline output

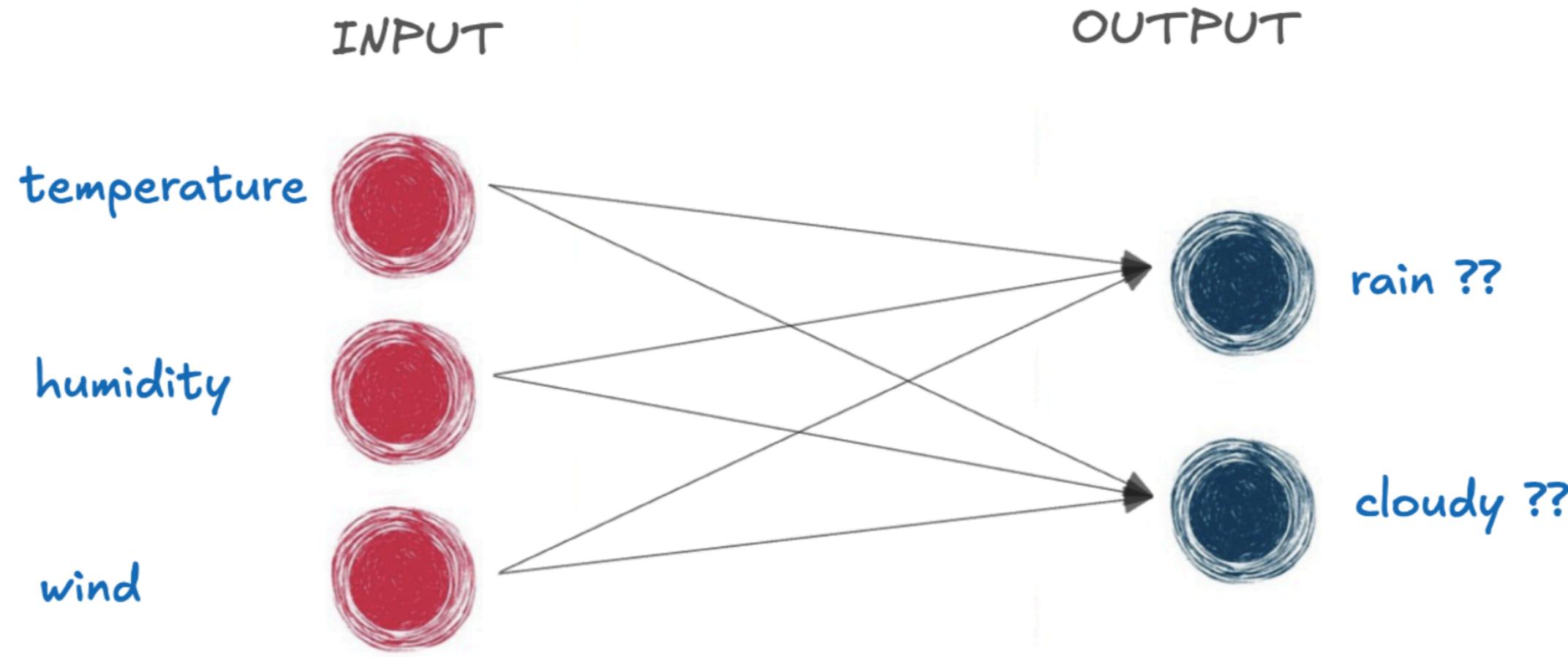
# A fully connected network in action



# A fully connected network in action



# A fully connected network in action



- Humidity feature will have a more significant **weight**
- Bias is to account for baseline information

# **Let's practice!**

**INTRODUCTION TO DEEP LEARNING WITH PYTORCH**

# Hidden layers and parameters

INTRODUCTION TO DEEP LEARNING WITH PYTORCH



Jasmin Ludolf

Senior Data Science Content Developer,  
DataCamp

# Stacking layers with nn.Sequential()

```
# Create network with three linear layers
model = nn.Sequential(
    nn.Linear(n_features, 8),
    nn.Linear(8, 4),
    nn.Linear(4, n_classes)
)
```

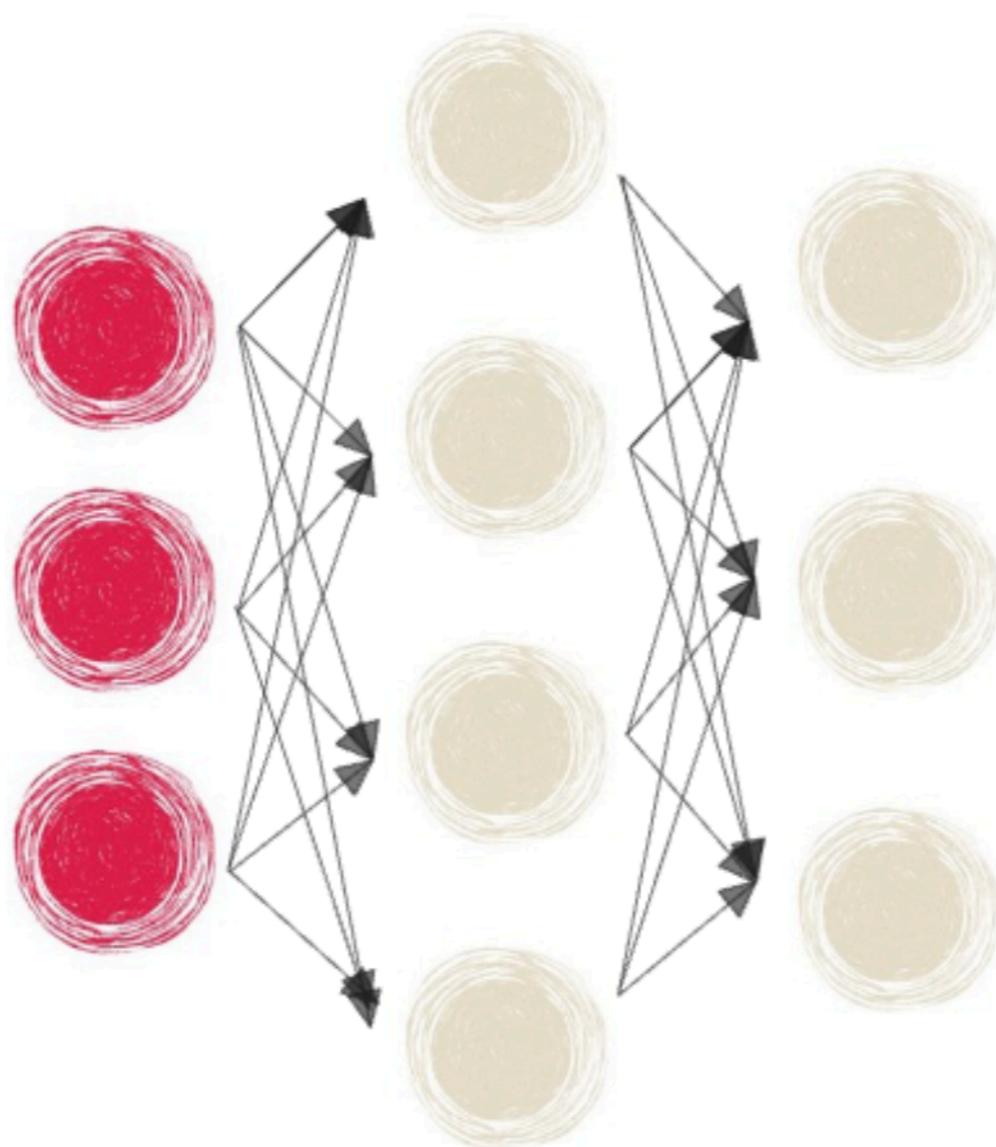
- Input is passed through the linear layers
- Layers within `nn.Sequential()` are hidden layers

# Stacking layers with nn.Sequential()

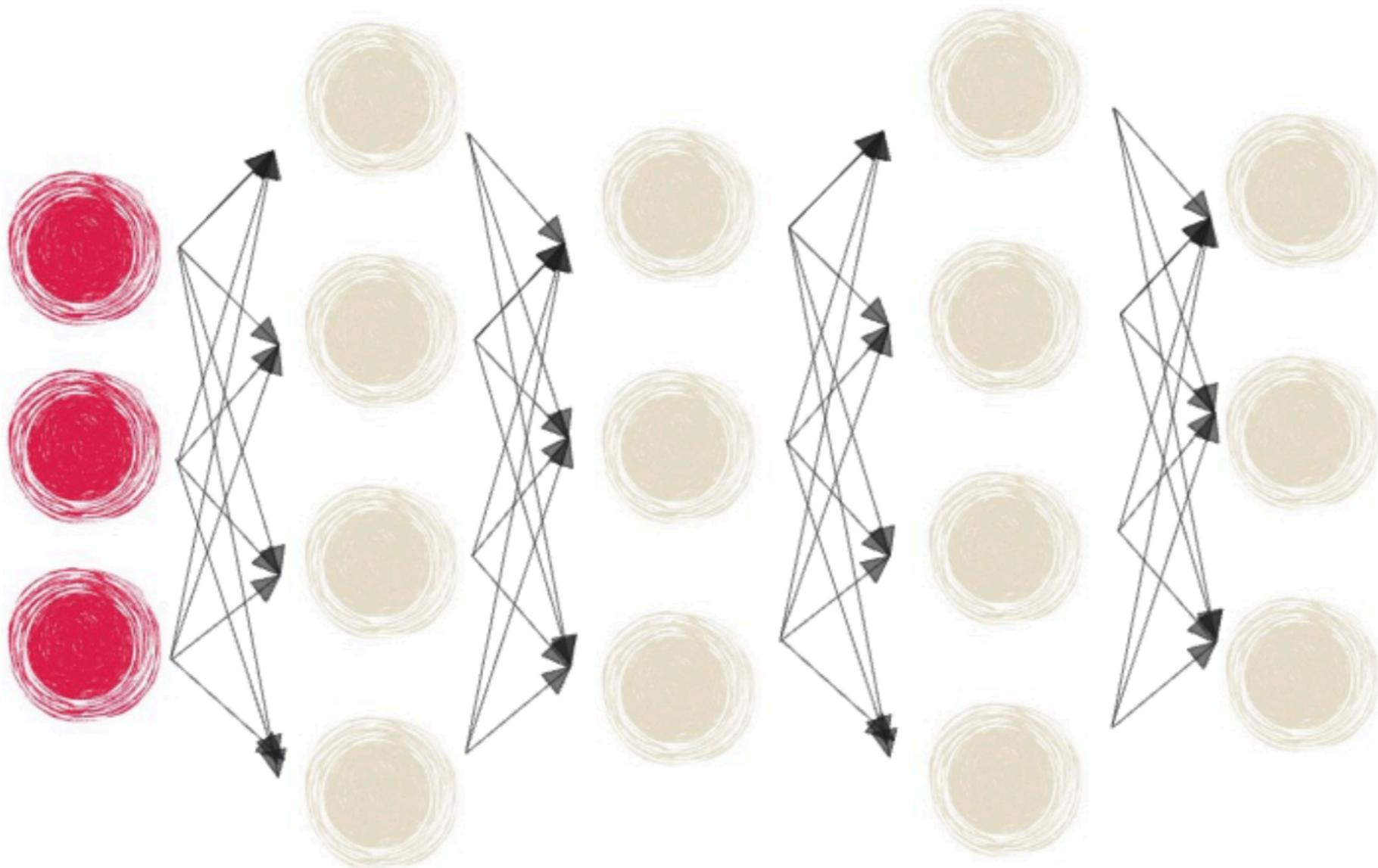
```
# Create network with three linear layers
model = nn.Sequential(
    nn.Linear(n_features, 8), # n_features represents number of input features
    nn.Linear(8, 4),
    nn.Linear(4, n_classes) # n_classes represents the number of output classes
)
```

- Input is passed through the linear layers
- Layers within `nn.Sequential()` are hidden layers
- `n_features` and `n_classes` are defined by the dataset

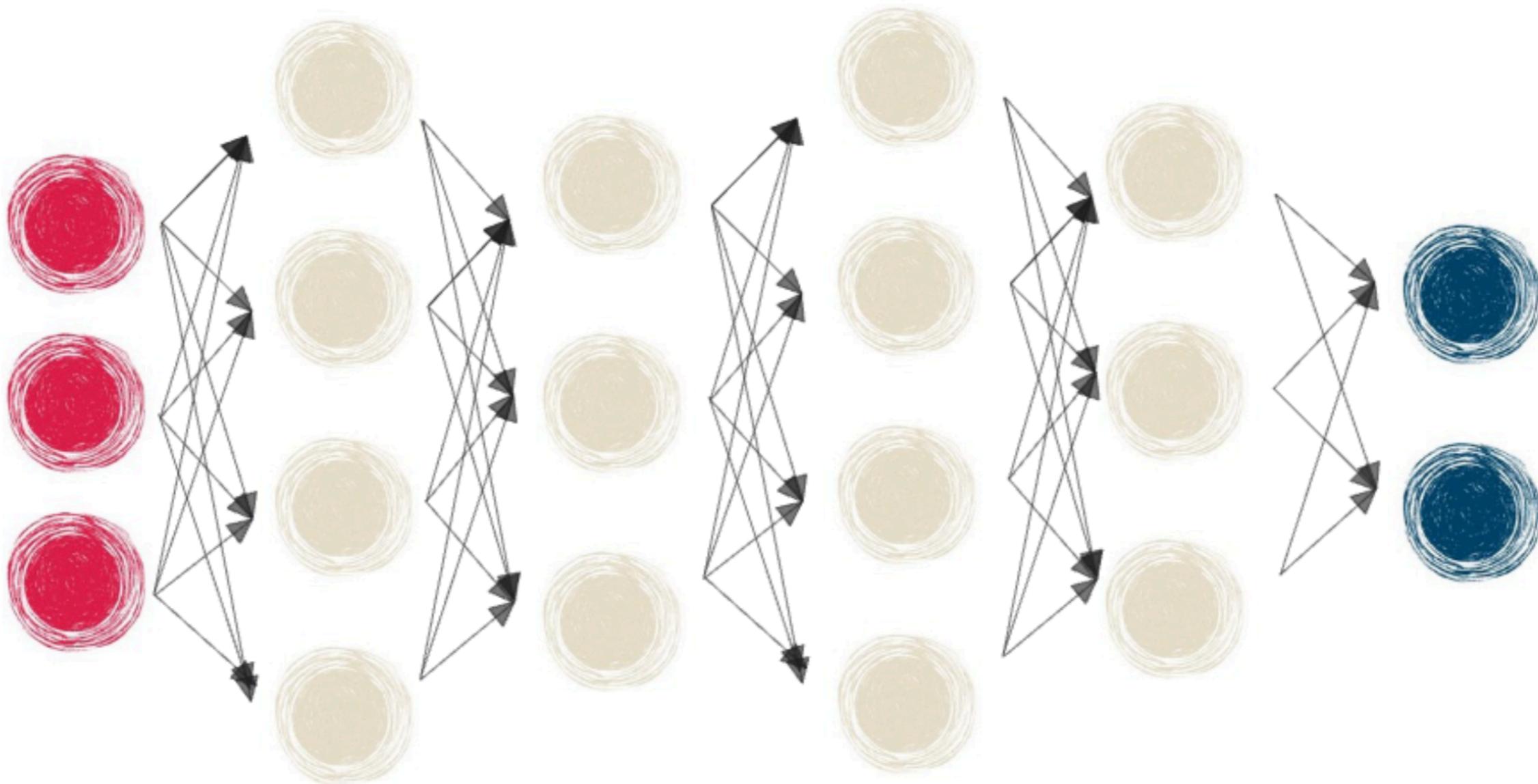
# Adding layers



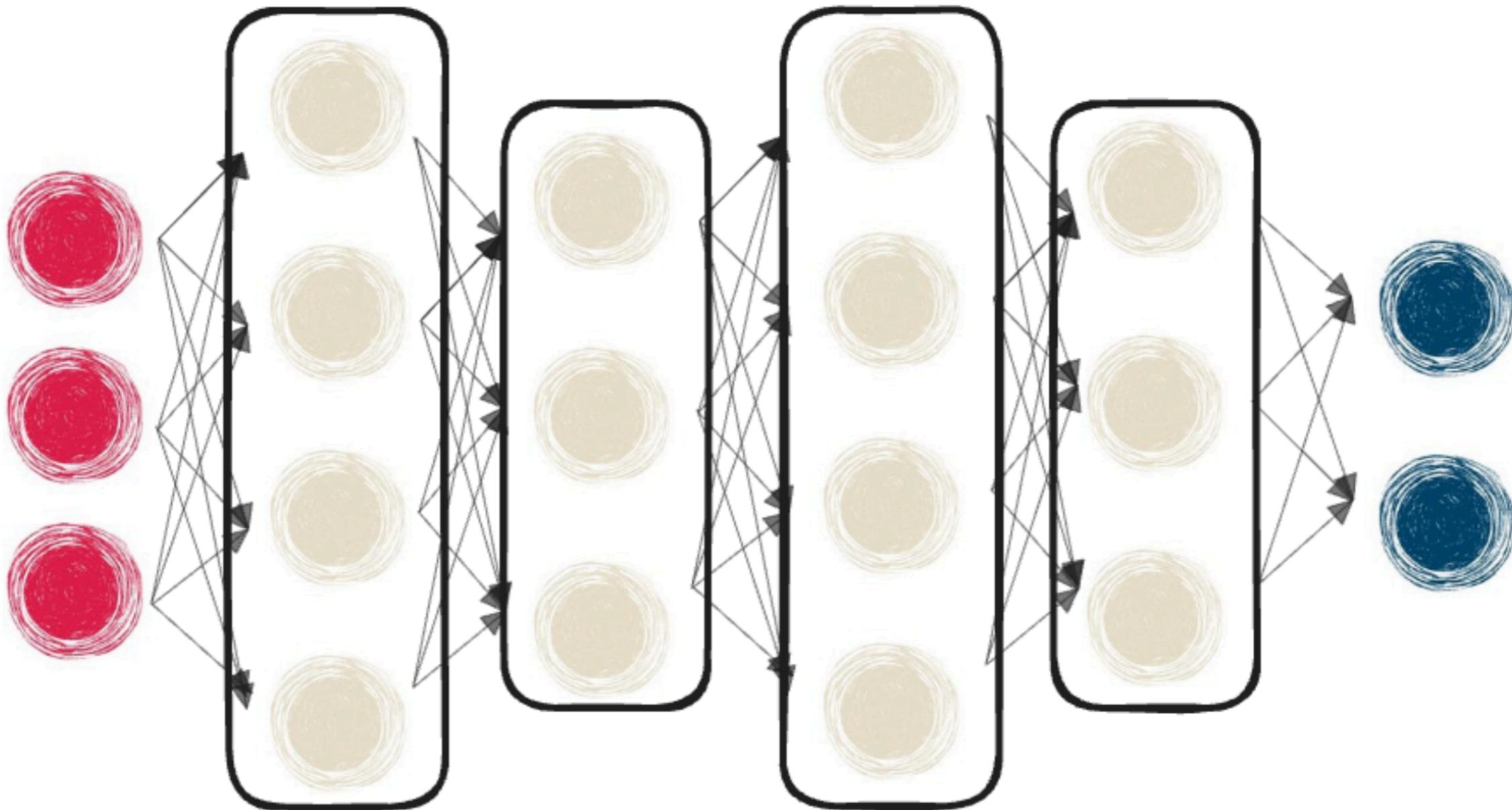
# Adding layers



# Adding layers



# Adding layers



# Adding layers

```
# Create network with three linear layers
model = nn.Sequential(
    nn.Linear(10, 18),
    nn.Linear(18, 20),
    nn.Linear(20, 5)
)
```

# Adding layers

```
# Create network with three linear layers
model = nn.Sequential(
    nn.Linear(10, 18), # Takes 10 features and outputs 18
    nn.Linear(18, 20),
    nn.Linear(20, 5)
)
```

- **Input 10 → output 18 → output 20 → Output 5**

# Adding layers

```
# Create network with three linear layers
model = nn.Sequential(
    nn.Linear(10, 18),
    nn.Linear(18, 20), # Takes 18 and outputs 20
    nn.Linear(20, 5)
)
```

- Input 10 → output 18 → output 20 → Output 5

# Adding layers

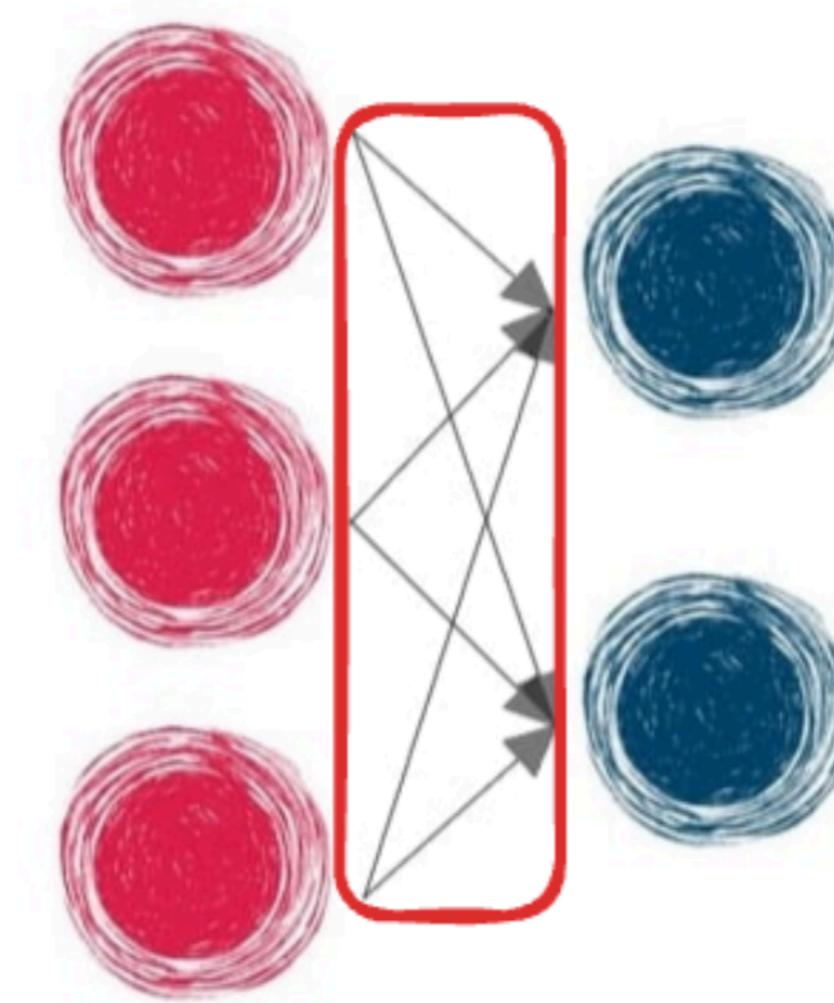
```
# Create network with three linear layers
model = nn.Sequential(
    nn.Linear(10, 18),
    nn.Linear(18, 20),
    nn.Linear(20, 5) # Takes 20 and outputs 5
)
```

- Input 10 → output 18 → output 20 → Output 5

# Layers are made of neurons

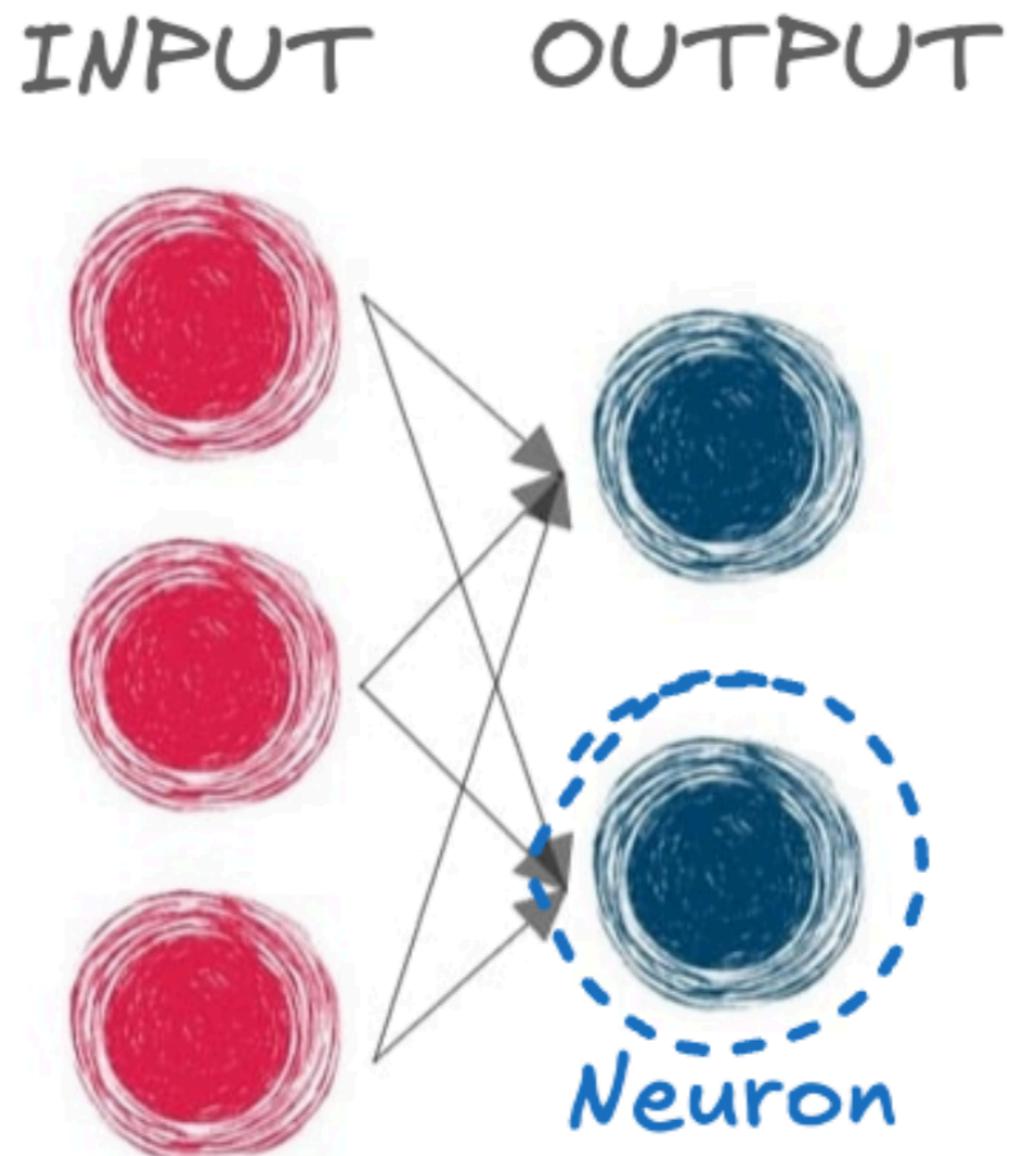
- Fully connected when each neuron links to all neurons in the previous layer

INPUT      OUTPUT



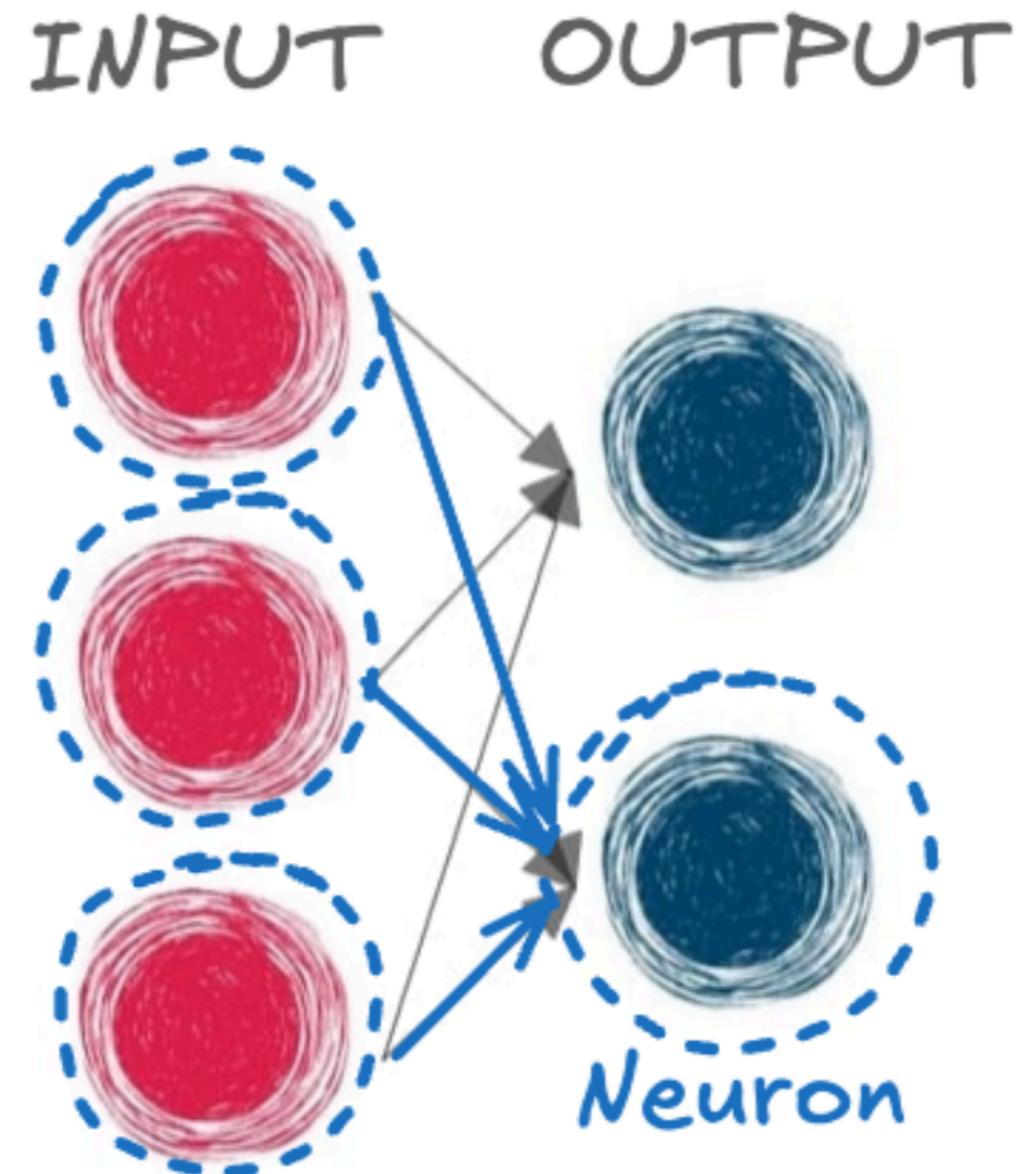
# Layers are made of neurons

- Fully connected when each neuron links to all neurons in the previous layer
- A neuron in a linear layer:



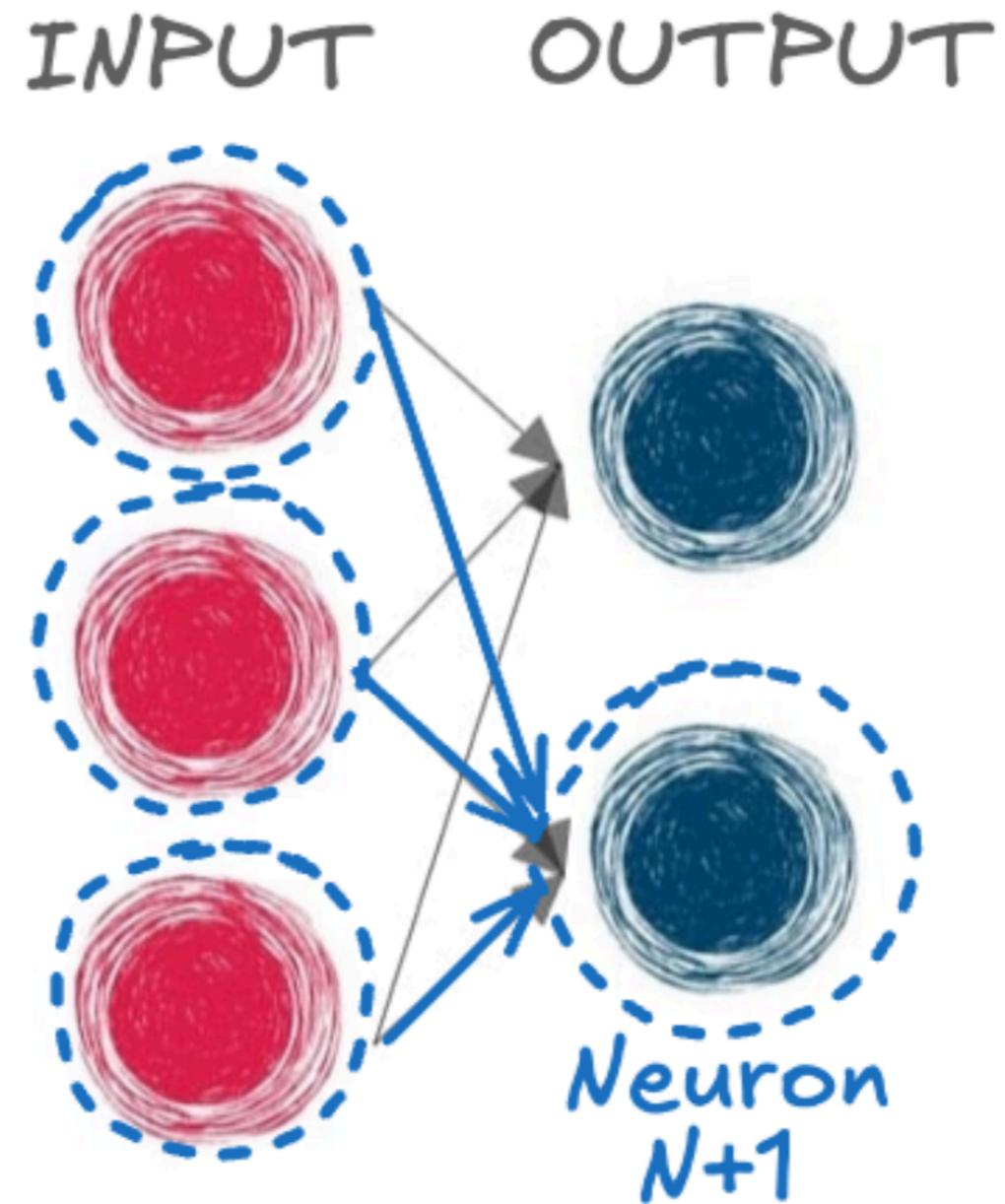
# Layers are made of neurons

- Fully connected when each neuron links to all neurons in the previous layer
- A neuron in a linear layer:
  - Performs a linear operation using **all neurons** from the previous layer



# Layers are made of neurons

- Fully connected when each neuron links to all neurons in the previous layer
- A neuron in a linear layer:
  - Performs a linear operation using **all neurons** from the previous layer
  - Has  $N+1$  parameters:  $N$  from **inputs** and 1 for the **bias**



# Parameters and model capacity

- More hidden layers = more parameters = higher **model capacity**

Given the following model:

```
model = nn.Sequential(nn.Linear(8, 4),  
                      nn.Linear(4, 2))
```

# Parameters and model capacity

- More hidden layers = more parameters = higher **model capacity**

Given the following model:

```
model = nn.Sequential(nn.Linear(8, 4),  
                      nn.Linear(4, 2))
```

**Manual parameter calculation:**

- First layer has 4 neurons, each neuron has 8+1 parameters. 9 times 4 = 36 parameters

# Parameters and model capacity

- More hidden layers = more parameters = higher **model capacity**

Given the following model:

```
model = nn.Sequential(nn.Linear(8, 4),  
                      nn.Linear(4, 2))
```

**Manual parameter calculation:**

- First layer has 4 neurons, each neuron has  $8+1$  parameters.  $9 \times 4 = 36$  parameters

# Parameters and model capacity

- More hidden layers = more parameters = higher **model capacity**

Given the following model:

```
model = nn.Sequential(nn.Linear(8, 4),  
                      nn.Linear(4, 2))
```

## Manual parameter calculation:

- First layer has 4 neurons, each neuron has  $8+1$  parameters.  $9 \times 4 = 36$  parameters
- Second layer has 2 neurons, each neuron has  $4+1$  parameters.  $5 \times 2 = 10$  parameters

# Parameters and model capacity

- More hidden layers = more parameters = higher **model capacity**

Given the following model:

```
model = nn.Sequential(nn.Linear(8, 4),  
                      nn.Linear(4, 2))
```

## Manual parameter calculation:

- First layer has 4 neurons, each neuron has  $8+1$  parameters.  $9 \times 4 = 36$  parameters
- Second layer has 2 neurons, each neuron has  $4+1$  parameters.  $5 \times 2 = 10$  parameters
- $36 + 10 = 46$  learnable parameters

# Parameters and model capacity

- More hidden layers = more parameters = higher **model capacity**

Given the following model:

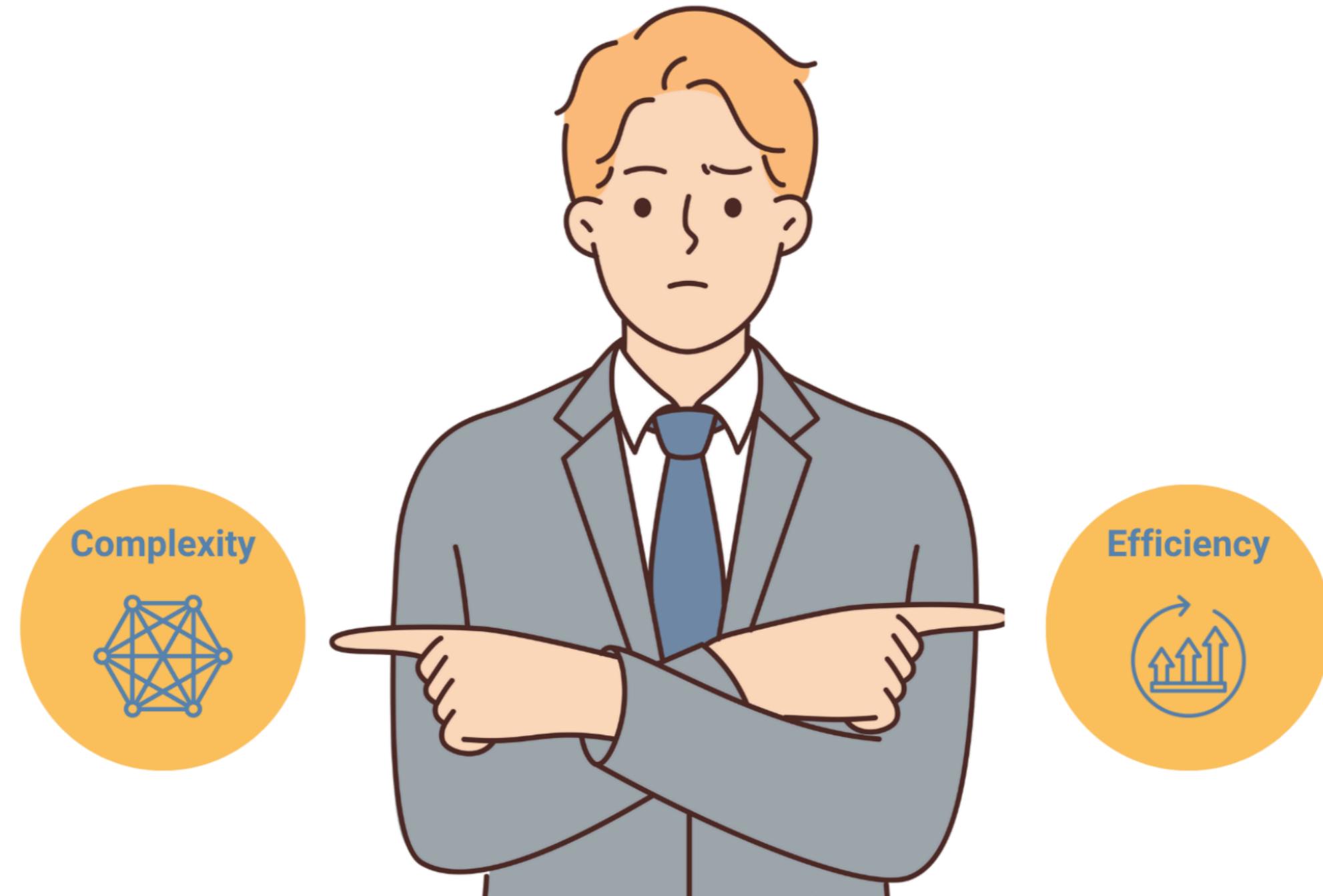
```
model = nn.Sequential(nn.Linear(8, 4),  
                      nn.Linear(4, 2))
```

Using PyTorch:

- `.numel()` : returns the number of elements in the tensor

```
total = 0  
for parameter in model.parameters():  
    total += parameter.numel()  
print(total)
```

# Balancing complexity and efficiency



# **Let's practice!**

**INTRODUCTION TO DEEP LEARNING WITH PYTORCH**