What is Keras?

INTRODUCTION TO DEEP LEARNING WITH KERAS

Theano vs Keras

```
import theano
import theano.tensor as T
from theano.ifelse import ifelse
import numpy as np
from random import random

# Define variables
x = T.matrix('x')
w1 = theano.shared(np.array([random(),random()]))
w2 = theano.shared(np.array([random(),random()]))
w3 = theano.shared(np.array([random(),random()]))
```

```
a2 = 1/(1+T.exp(-T.dot(x,w2)-b1))
x2 = T.stack([a1,a2],axis=1)
a3 = 1/(1+T.exp(-T.dot(x2,w3)-b2))

a_hat = T.vector('a_hat') #Actual output
cost = -(a_hat*T.log(a3) + (1-a_hat)*T.log(1-a3)).sum()
dw1,dw2,dw3,db1,db2 = T.grad(cost,[w1,w2,w3,b1,b2])
```

```
on
```

```
        [w1, w1-learning_rate*dw1],
        [w2, w2-learning_rate*dw2],
        [w3, w3-learning_rate*dw3],
        [b1, b1-learning_rate*db1],
        [b2, b2-learning_rate*db2]
    ]
```

```
# You can (finally) train your model
cost = []
for iteration in range(30000):
    pred, cost_iter = train(inputs, outputs)
    cost.append(cost_iter)
```

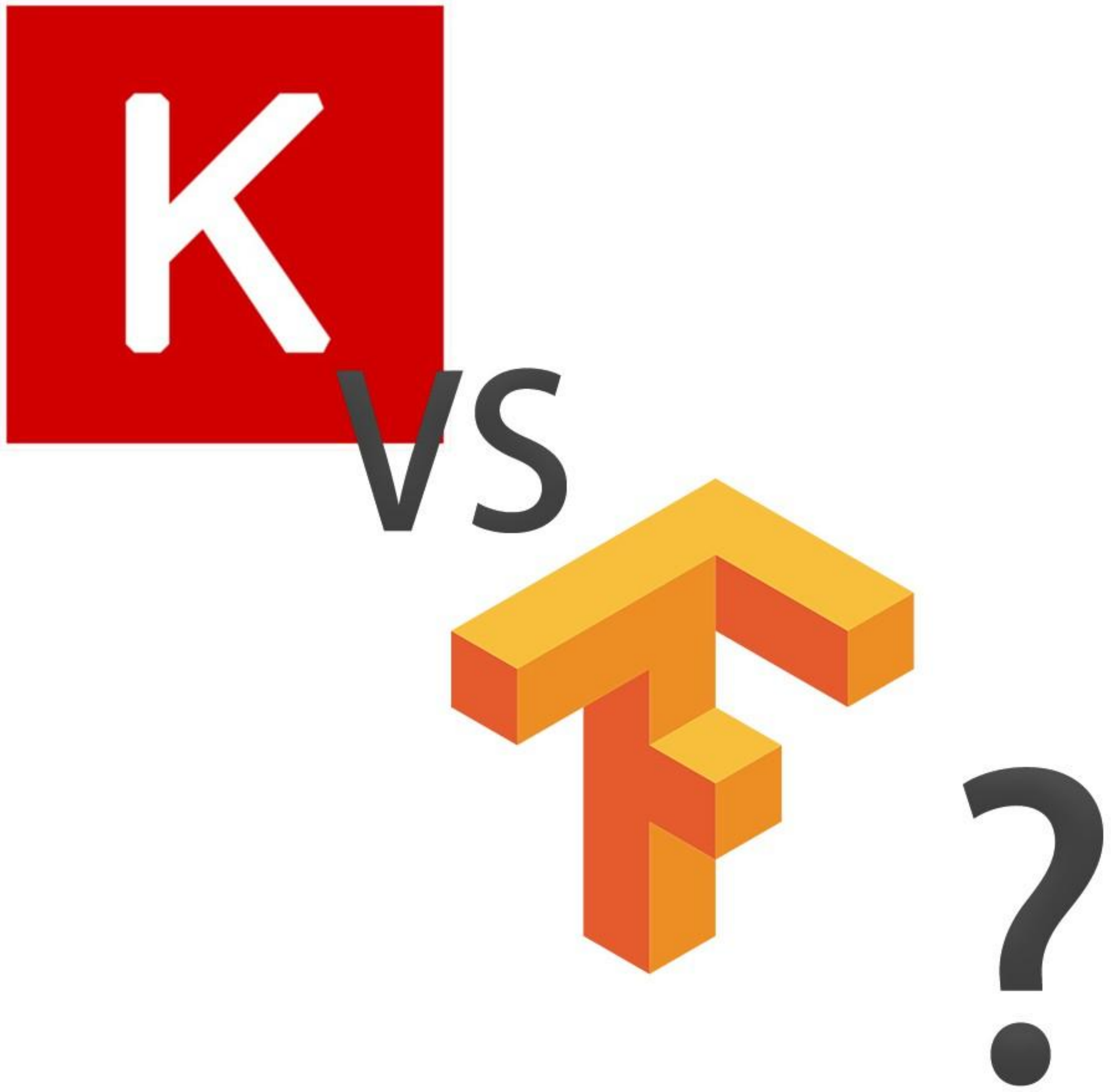
```
from keras.layers import Dense
from keras.models import Sequential

# Define model and add layers
model = Sequential()
model.add(Dense(2,input_shape=(2,),activation='sigmoid'))
model.add(Dense(1,activation='sigmoid'))

model.compile(optimizer='adam',loss='categorical_crossentropy')

# Train model
model.fit(inputs,outputs)
```

Keras

- Deep Learning Framework
- Enables fast experimentation
- Runs on top of other frameworks
- Wri en by François Chollet

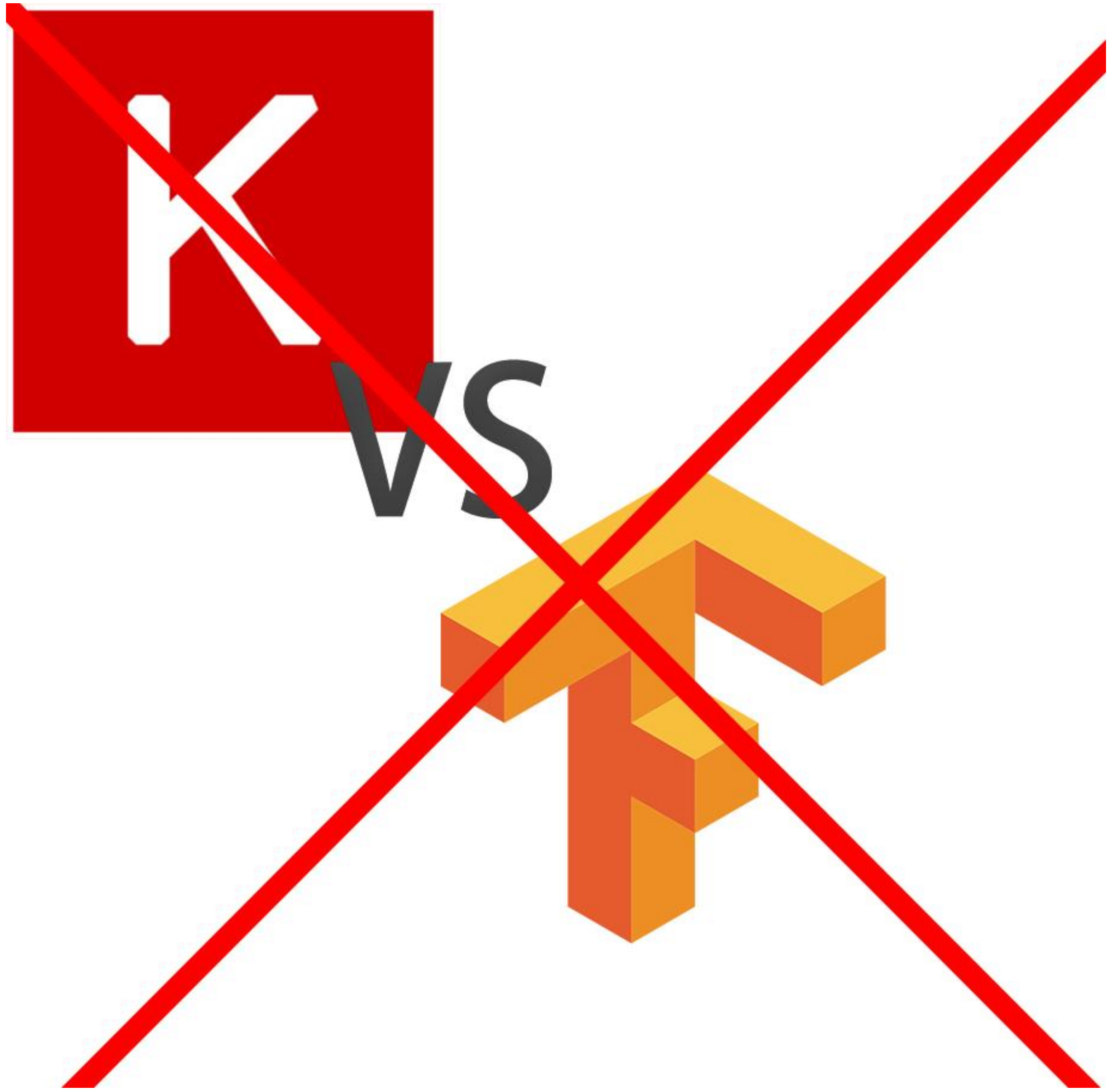
Why use Keras?

- Fast industry-ready models
- For beginners and experts
- Less code
- Build any architecture
- Deploy models in multiple platforms



Keras + TensorFlow

- TensorFlow's high level framework of choice
- Keras is complementary to TensorFlow
- You can use TensorFlow for low level features



1. Level of API

	Keras	PyTorch
High- and Low-Level API	High Level API	Low Level API

2. Speed

	Keras	PyTorch
Very Fast, used for high performance	Slower than TensorFlow as it works on top of TensorFlow	Same speed as TensorFlow
		

3. Architecture

	Keras	PyTorch
Has a complex architecture and is hard to use	Has a simpler architecture as abstraction is used to make it simple to use	Has a complex architecture
		

4. Datasets and Debugging

Keras

PyTorch

Used for very high-performance models. Debugging is hard

Used for smaller datasets. Debugging is easy and less frequent due to smaller models

Used for large datasets. Easier to debug than TensorFlow

5. Ease of Development

	Keras	PyTorch
Hard to develop and write code	Easy to develop and is best for newbies	Easier to learn than TensorFlow

6. Ease of Deployment

	Keras	PyTorch
Easy to deploy with 'TensorFlow Serving'	Model deployment can be done with TensorFlow serving or Flask	'Pytorch Mobile' makes deployment easy, but not as much as in TensorFlow

Which framework should you use?

TensorFlow has implemented various levels of abstraction to make implementation easy. This also makes debugging easy

Keras

It is simple and easy, but not as fast as TensorFlow.
It is more user-friendly than any other deep learning API

PyTorch

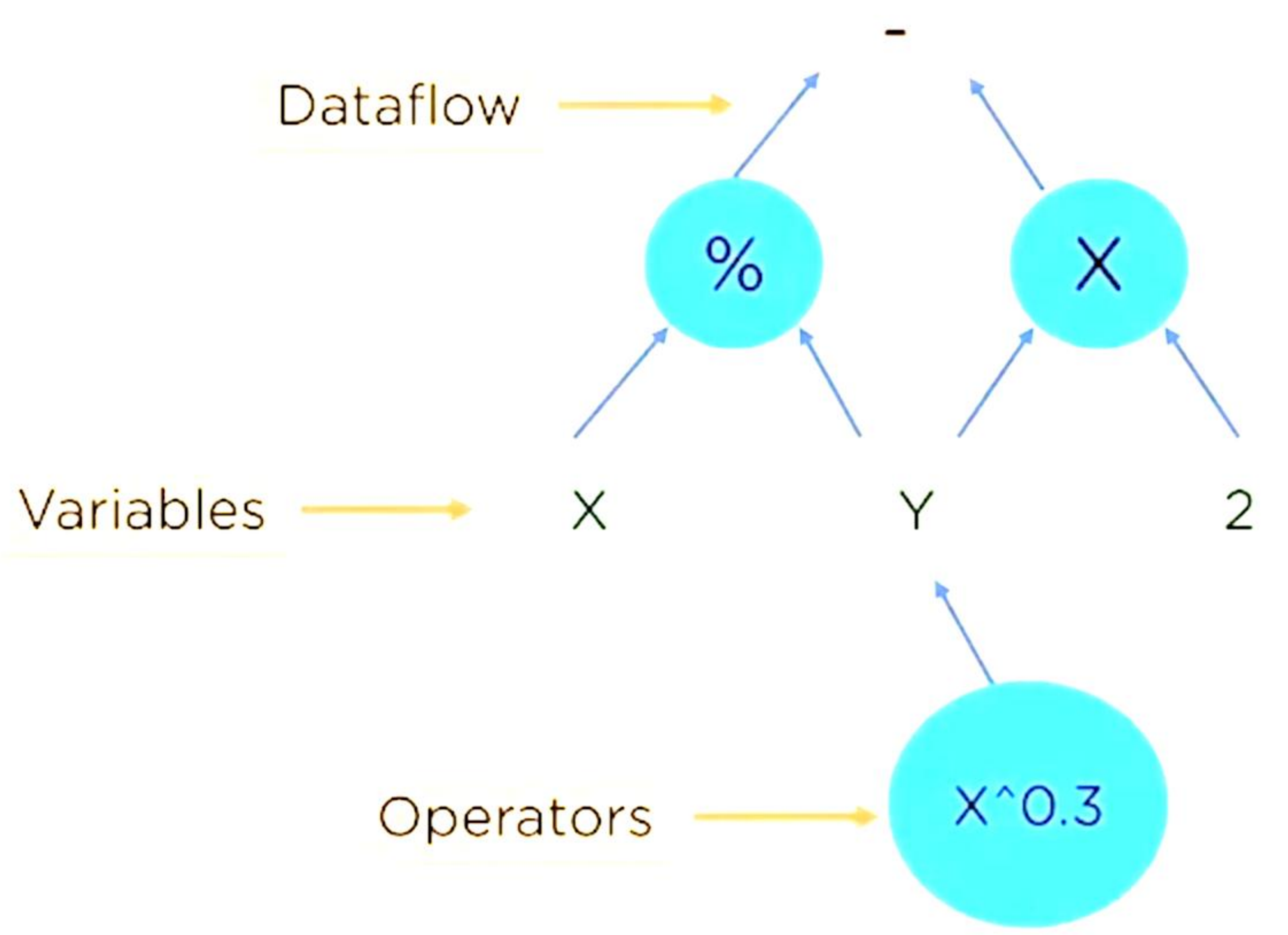
It is the preferred deep learning API for teachers but is not as widely used in production as TensorFlow. Faster, but lower GPU utilization

Working principle of Keras

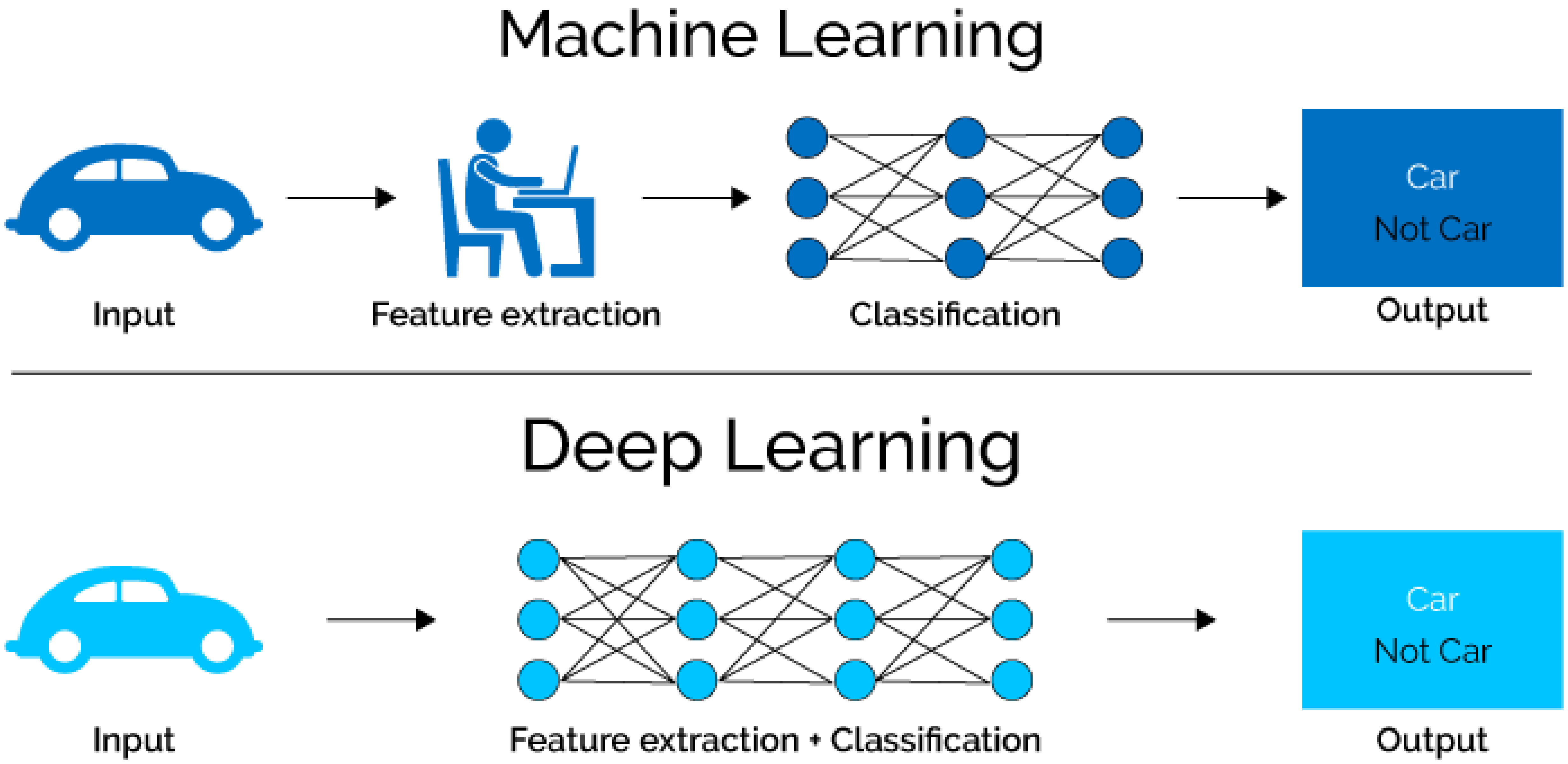
Keras uses computational graphs to express and evaluate mathematical expressions

1. Expressing complex problems as combination of simple mathematical operators

2. Useful for calculating derivatives by using backpropagation

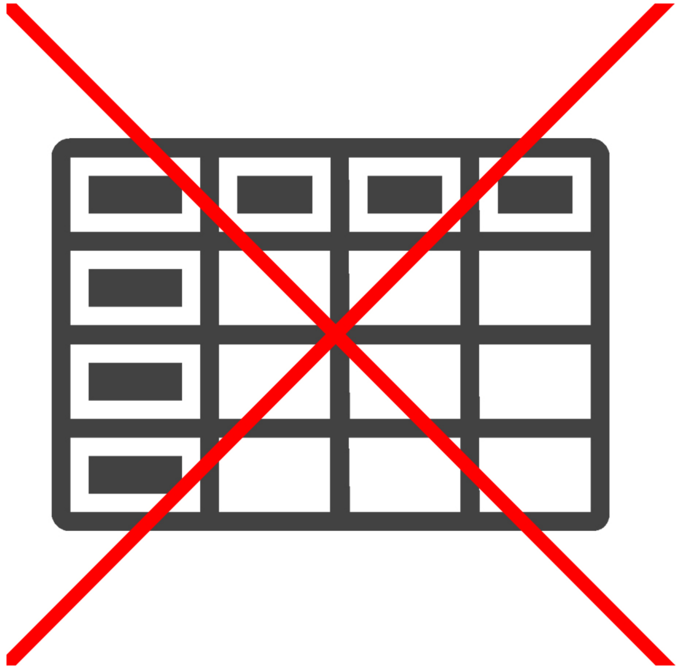


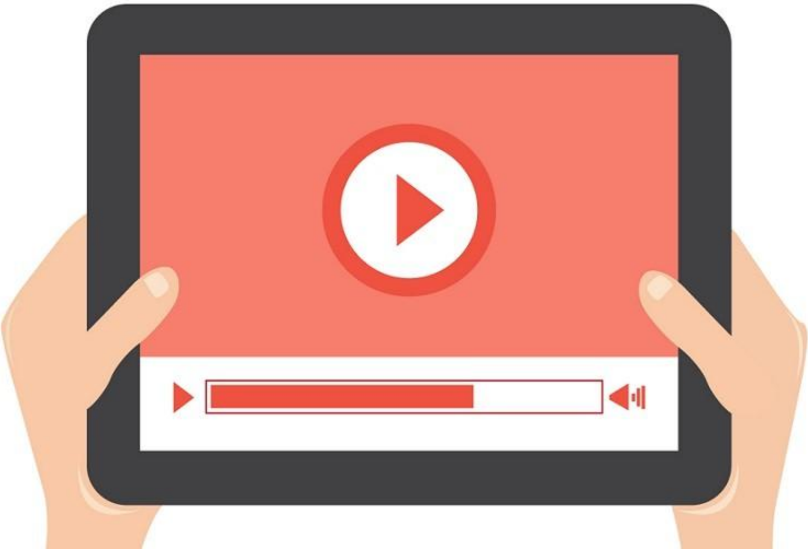
Feature Engineering



[1] Towards Data Science

Unstructured data



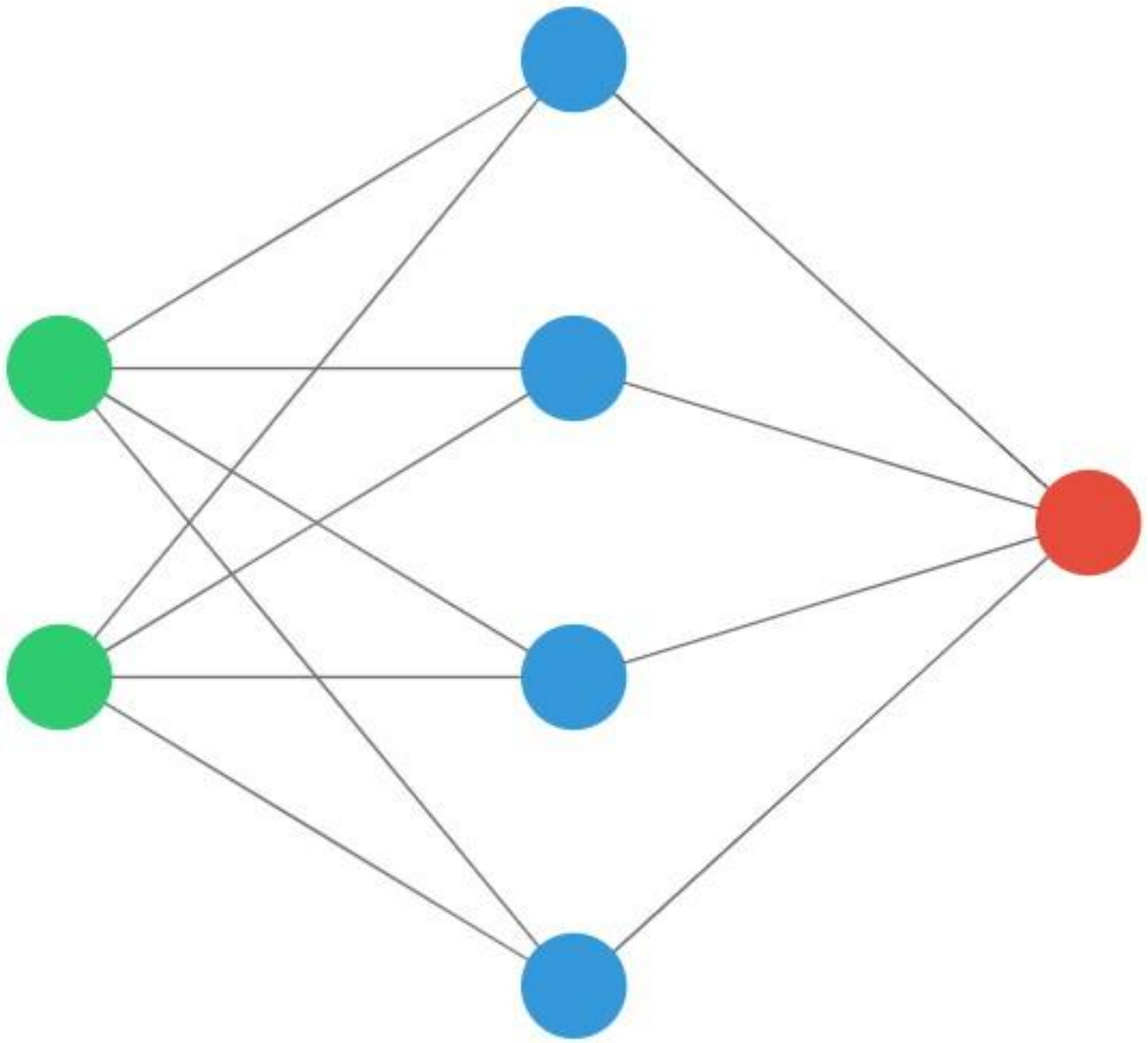


So, when to use neural networks?

- Dealing with unstructured data
- Don't need easily interpretable results
- You can bene t from a known architecture

Example: Classify images of cats and dogs

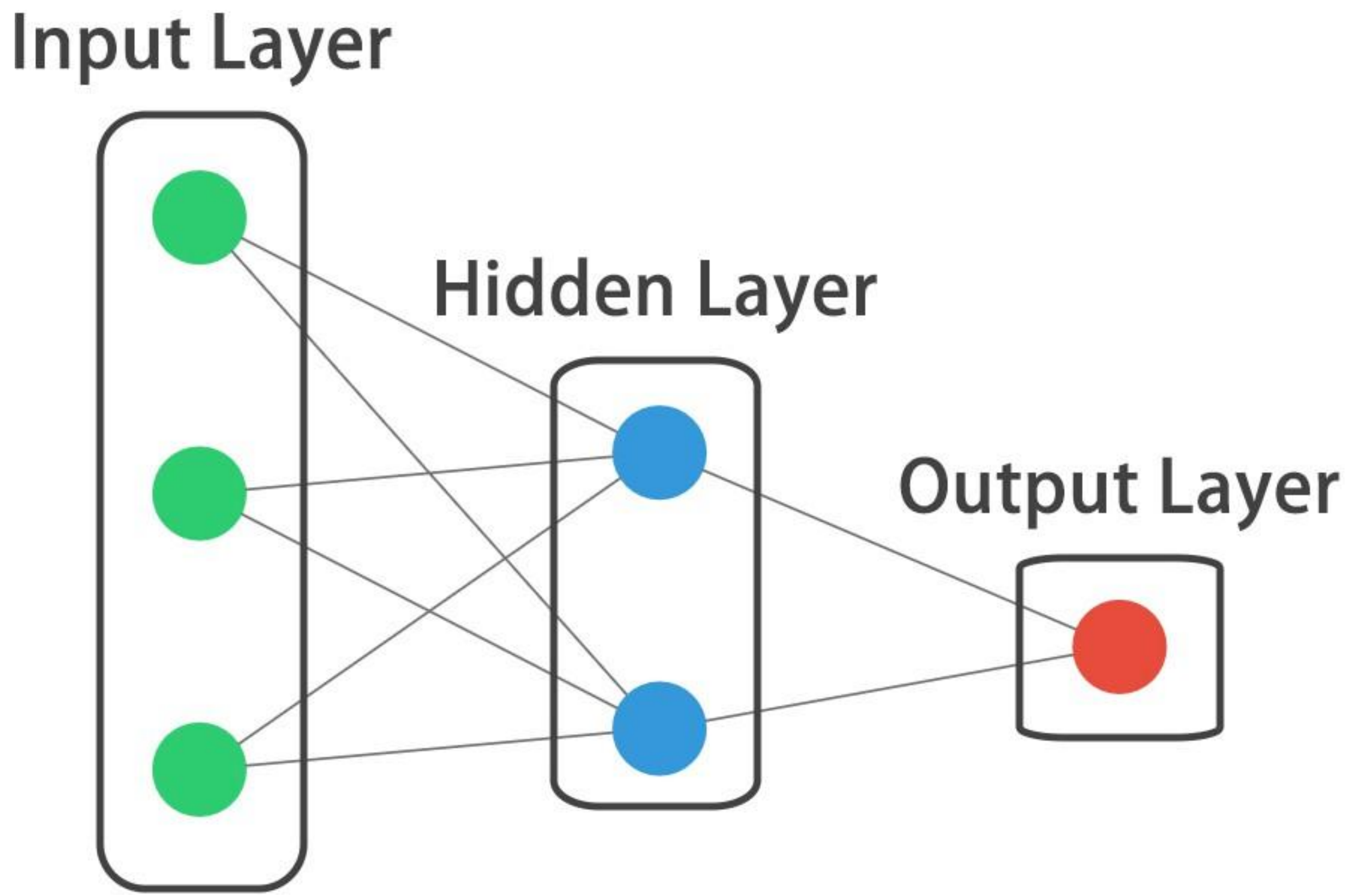
- Images -> Unstructured data
- You don't care about why the network knows it's a cat or a dog
- You can bene t from convolutional neural networks



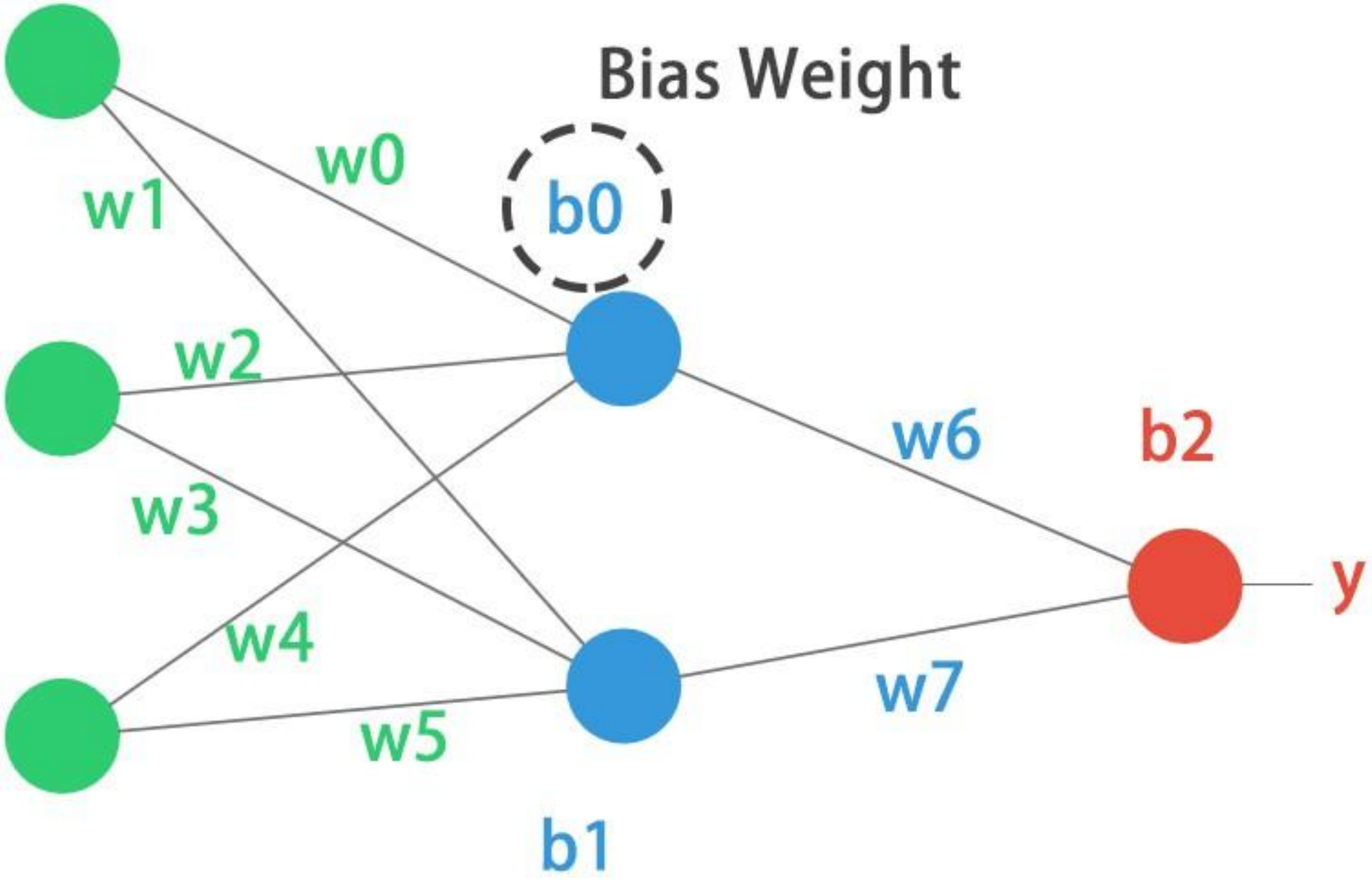
Your first neural network

INTRODUCTION TO DEEP LEARNING WITH KERAS

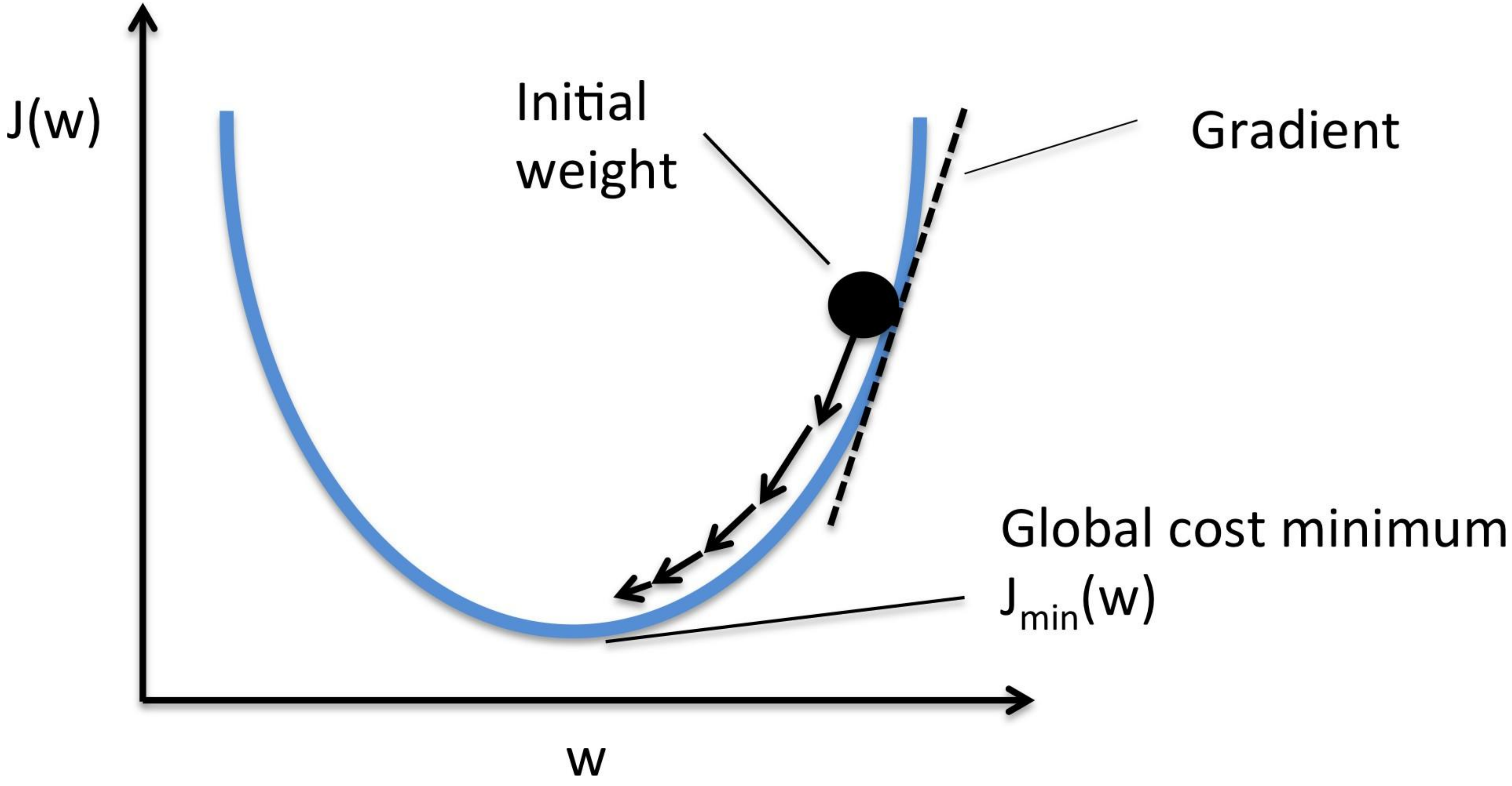
A neural network?



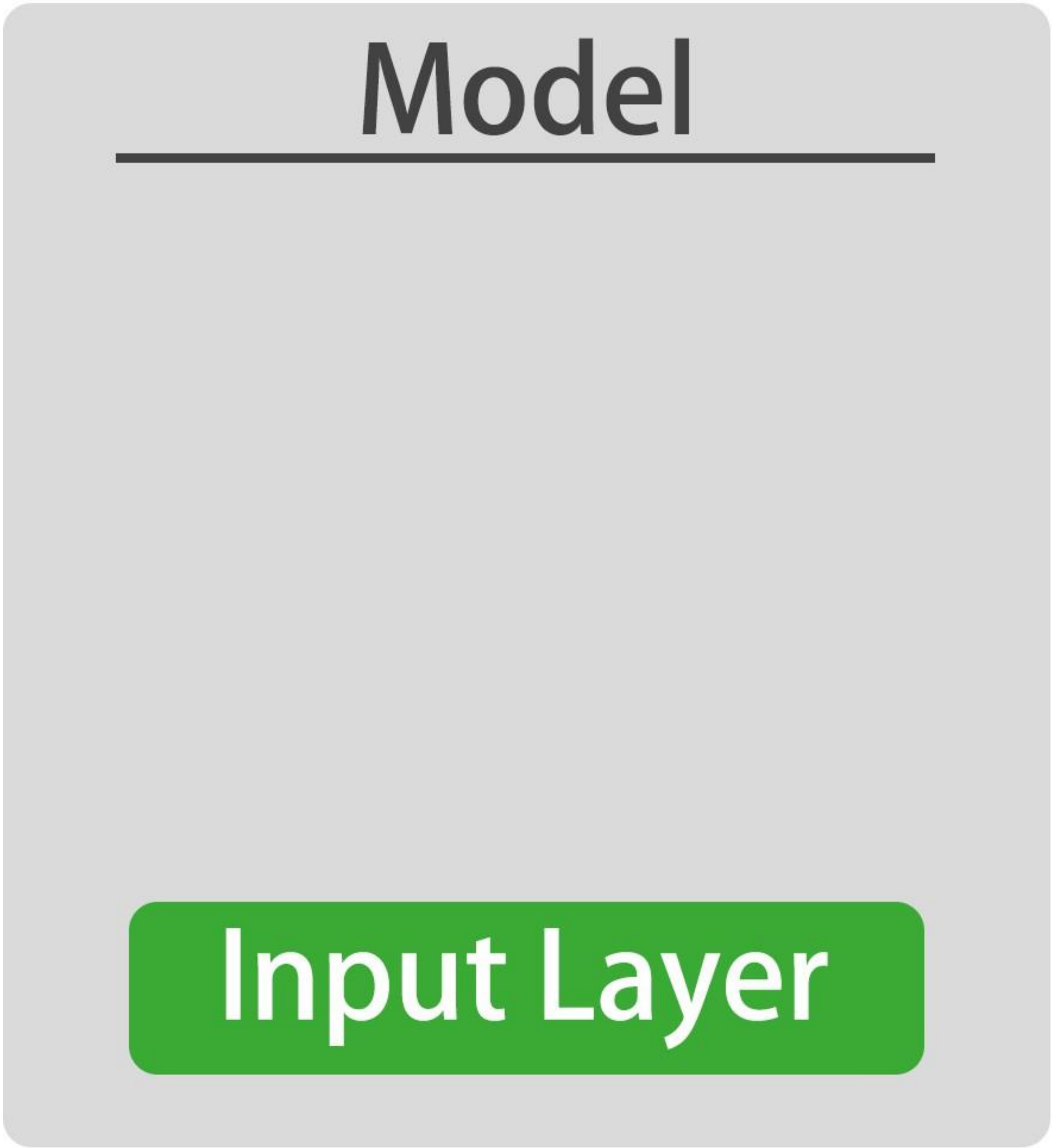
Parameters



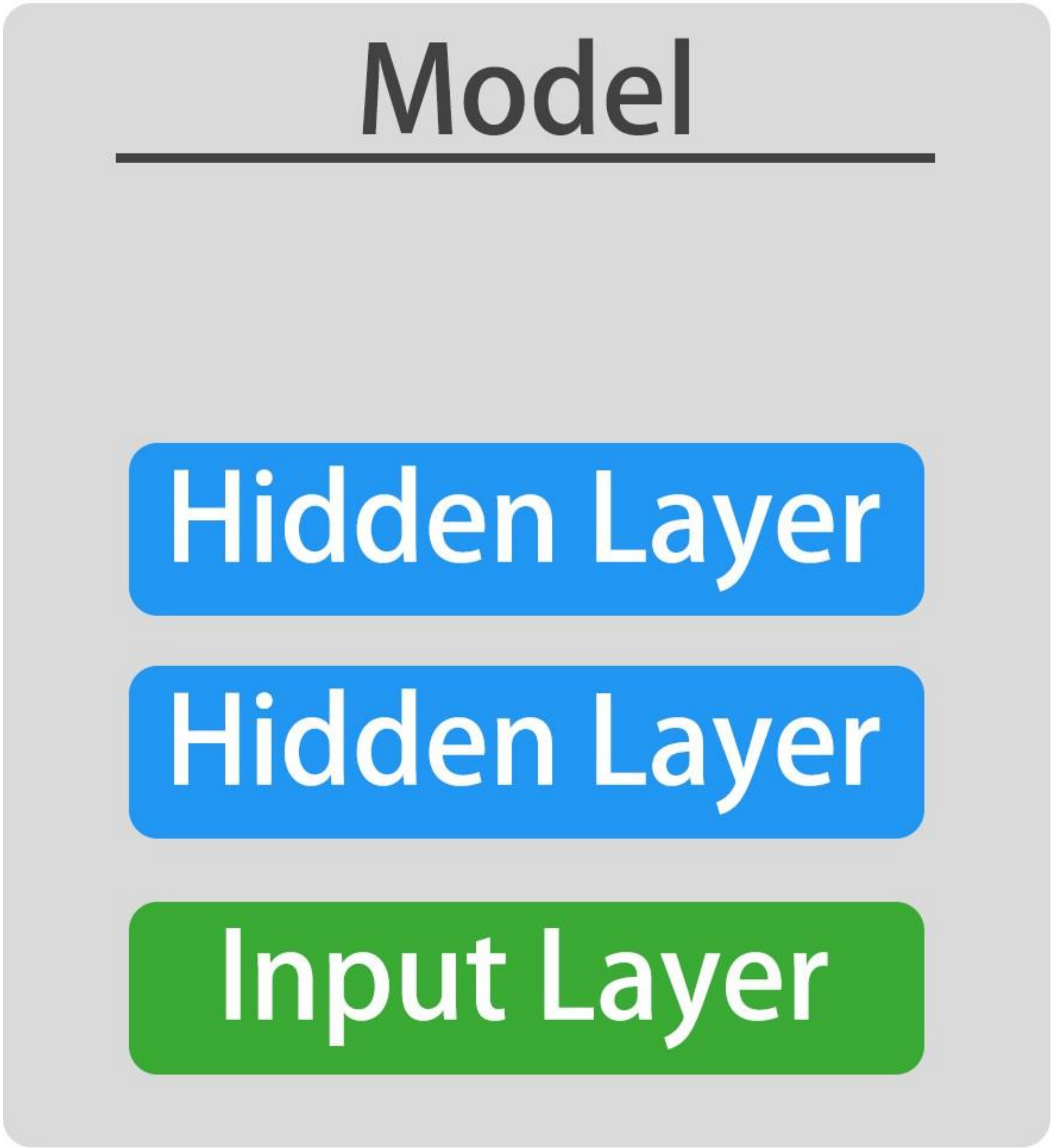
Gradient descent



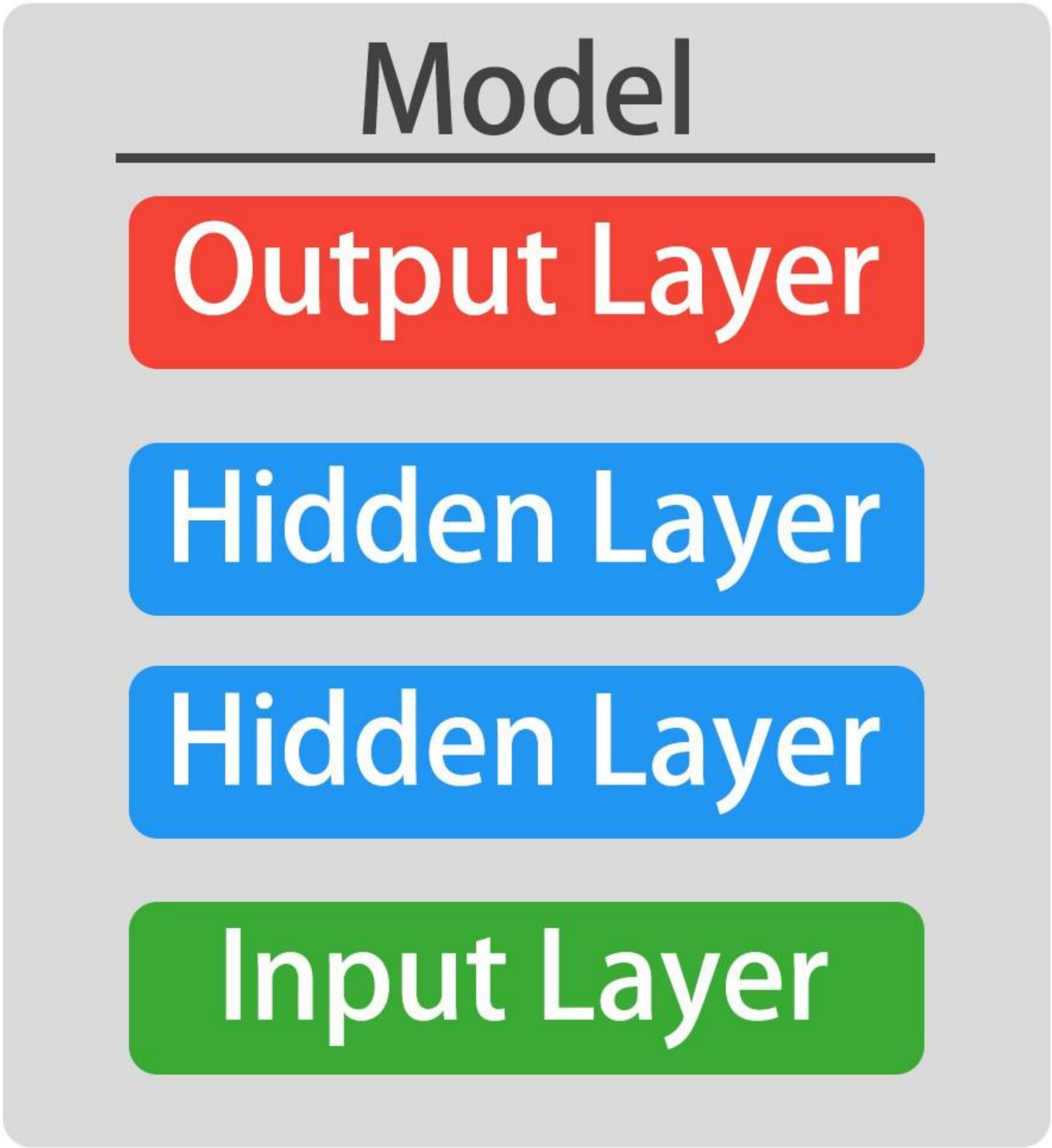
The sequential API



The sequential API

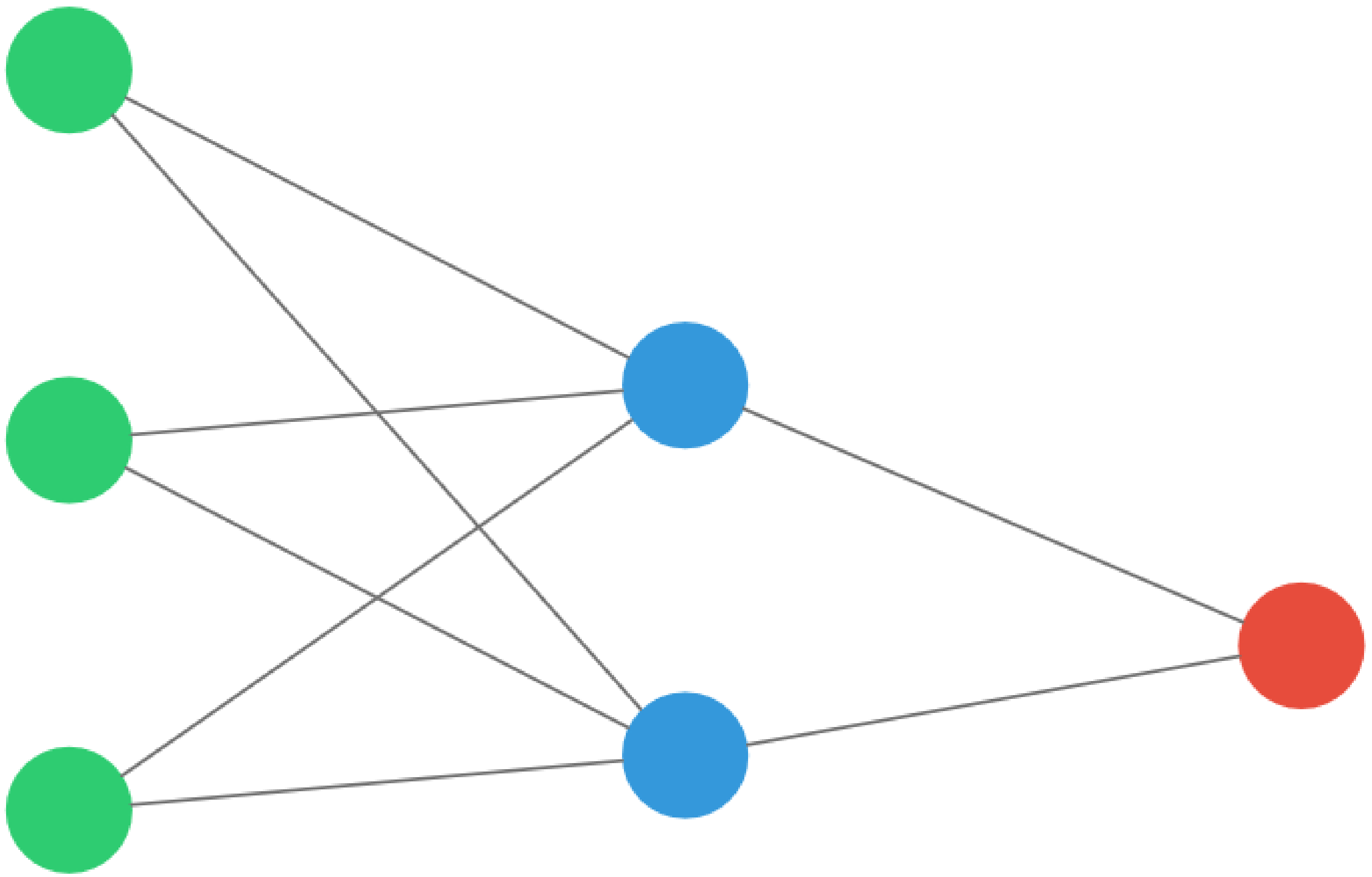


The sequential API



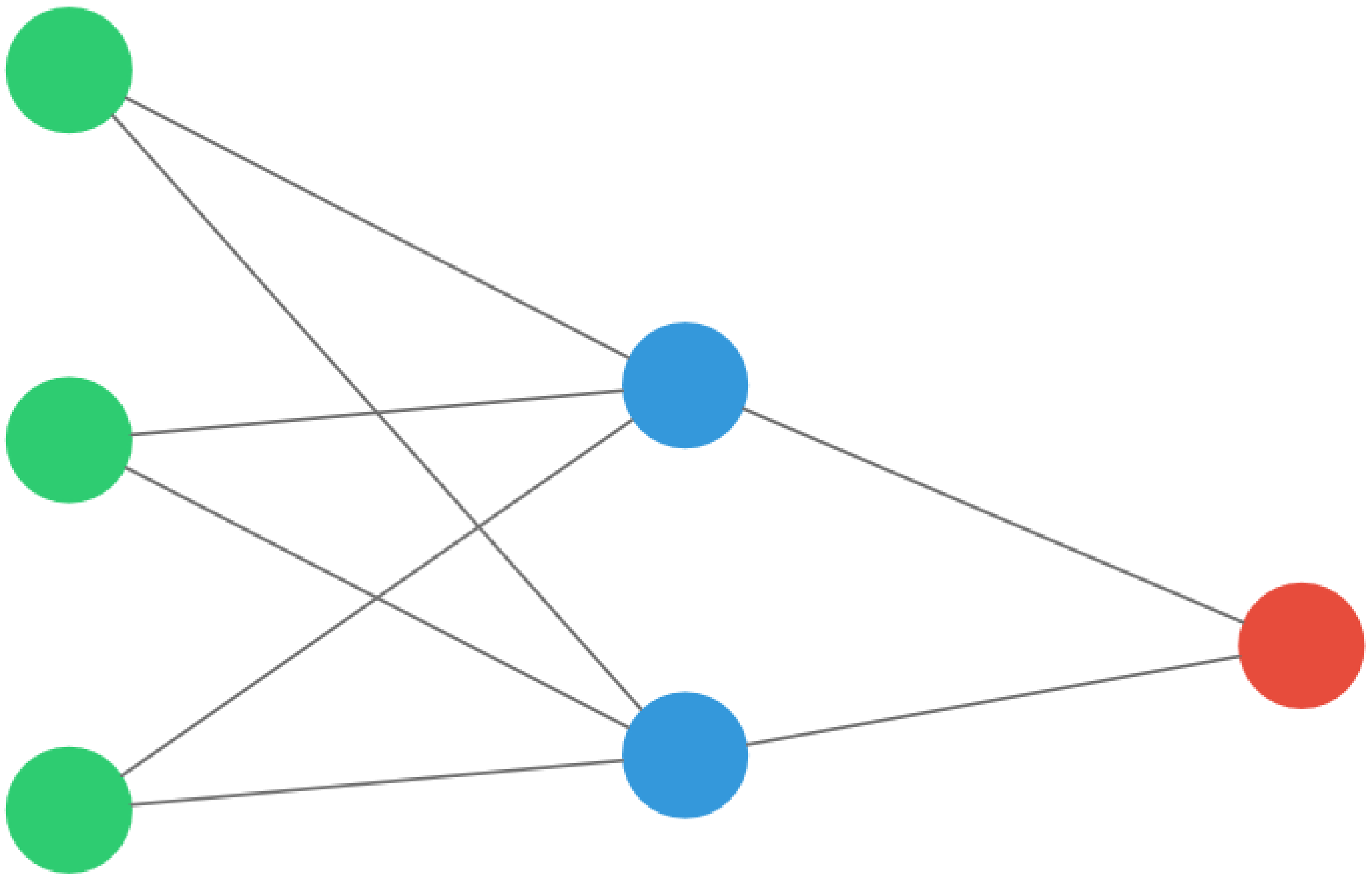
Defining a neural network

```
from tensorflow.keras.models import Sequ
from tensorflow.keras.layers import Dens
# Create a new sequential model
model = Sequential()
# Add and input and dense layer
model.add(Dense(2, input_shape=(3,)))
# Add a final 1 neuron layer
model.add(Dense(1))
```



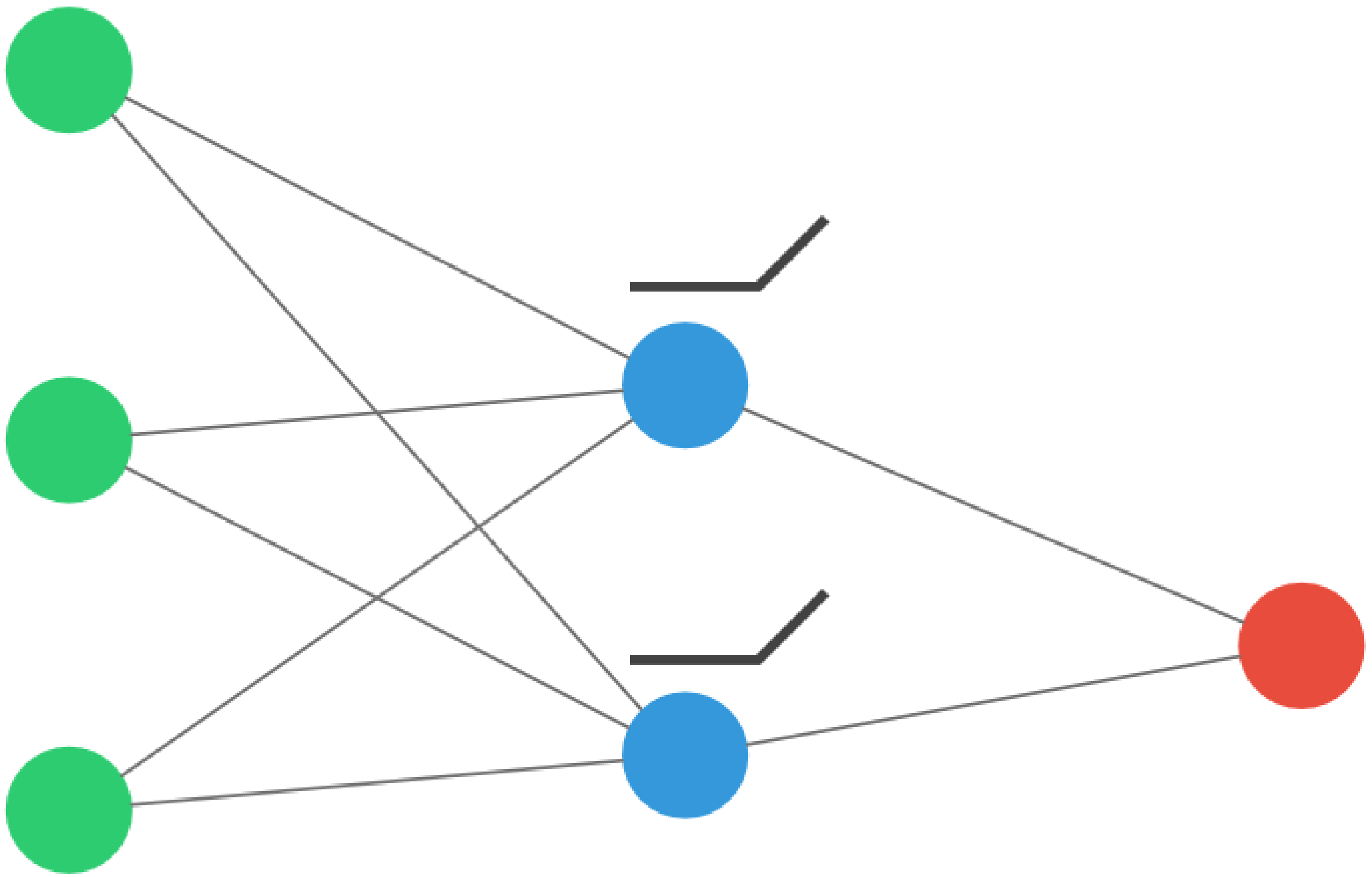
Adding activations

```
from tensorflow.keras.models import Sequ
from tensorflow.keras.layers import Dens
# Create a new sequential model
model = Sequential()
# Add and input and dense layer
model.add(Dense(2, input_shape=(3,)))
# Add a final 1 neuron layer
model.add(Dense(1))
```



Adding activations

```
from tensorflow.keras.models import Sequ
from tensorflow.keras.layers import Dens
# Create a new sequential model
model = Sequential()
# Add and input and dense layer
model.add(Dense(2, input_shape=(3,),
                activation="relu"))
# Add a final 1 neuron layer
model.add(Dense(1))
```

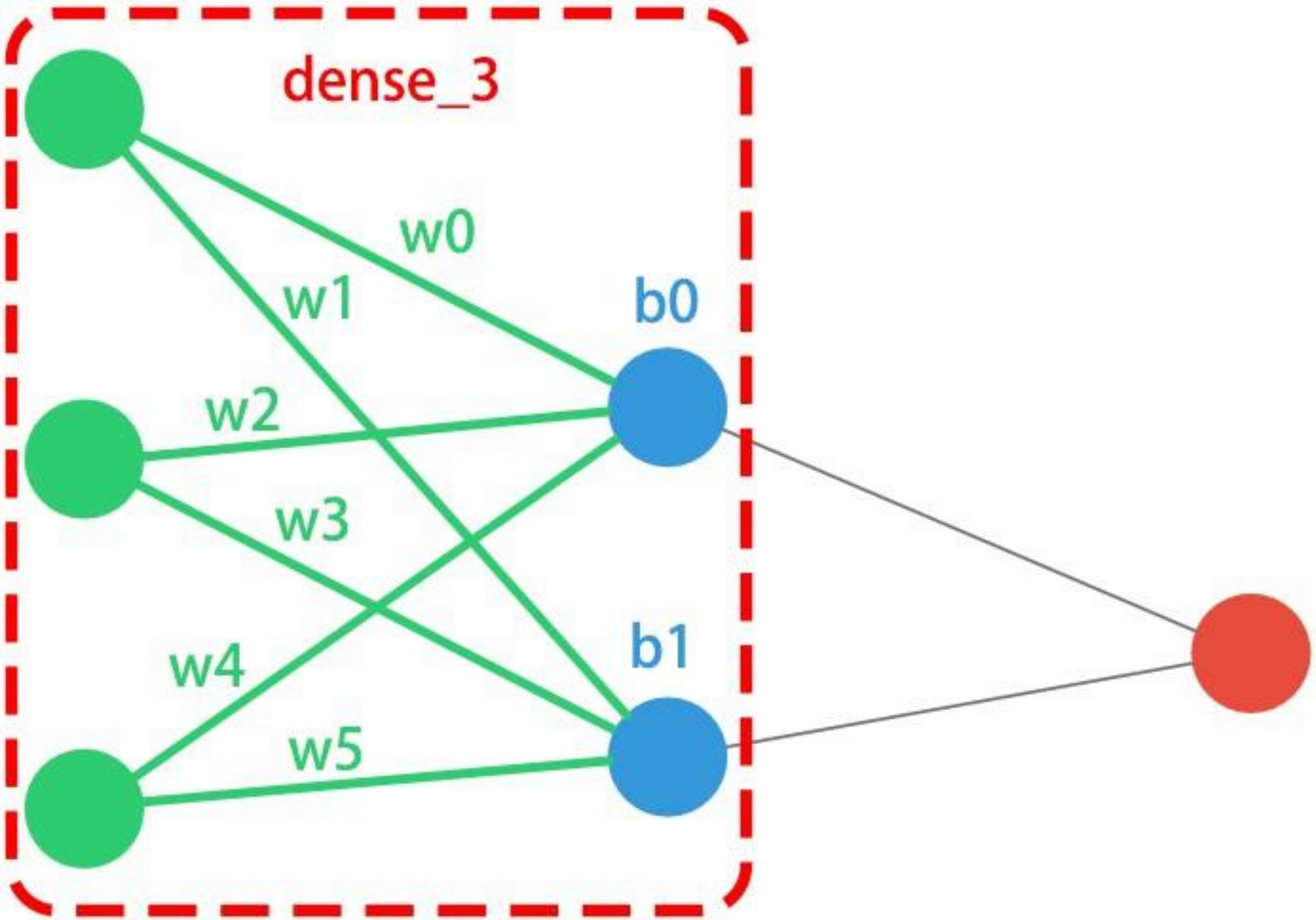


Summarize your model!

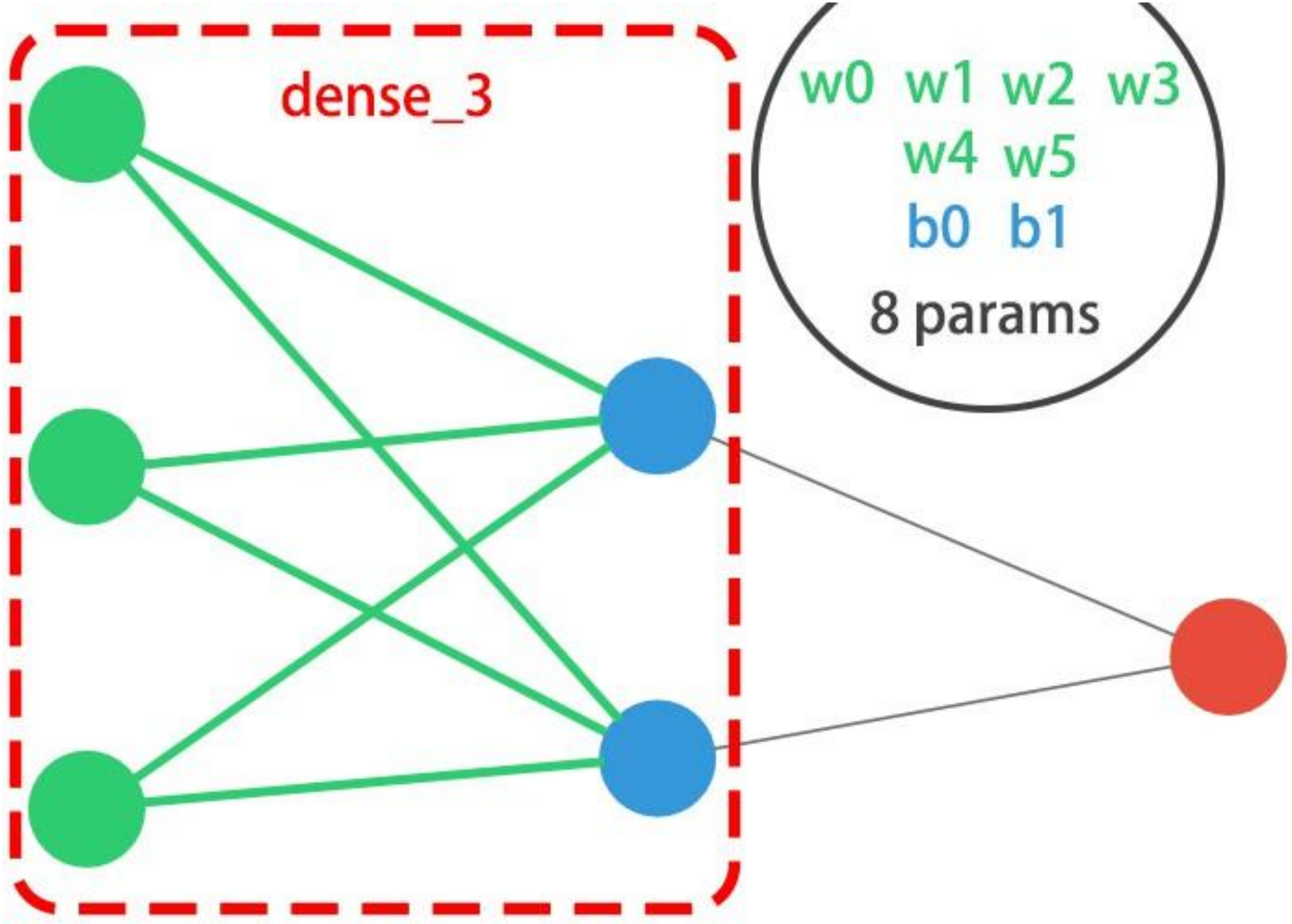
```
model.summary()
```

```
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)               (None, 2)                 8
_________________________________________________________________
dense_4 (Dense)               (None, 1)                 3
=================================================================
Total params: 11
Trainable params: 11
Non-trainable params: 0
_________________________________________________________________
```

Visualize parameters



Visualize parameters



Summarize your model!

```
model.summary()
```

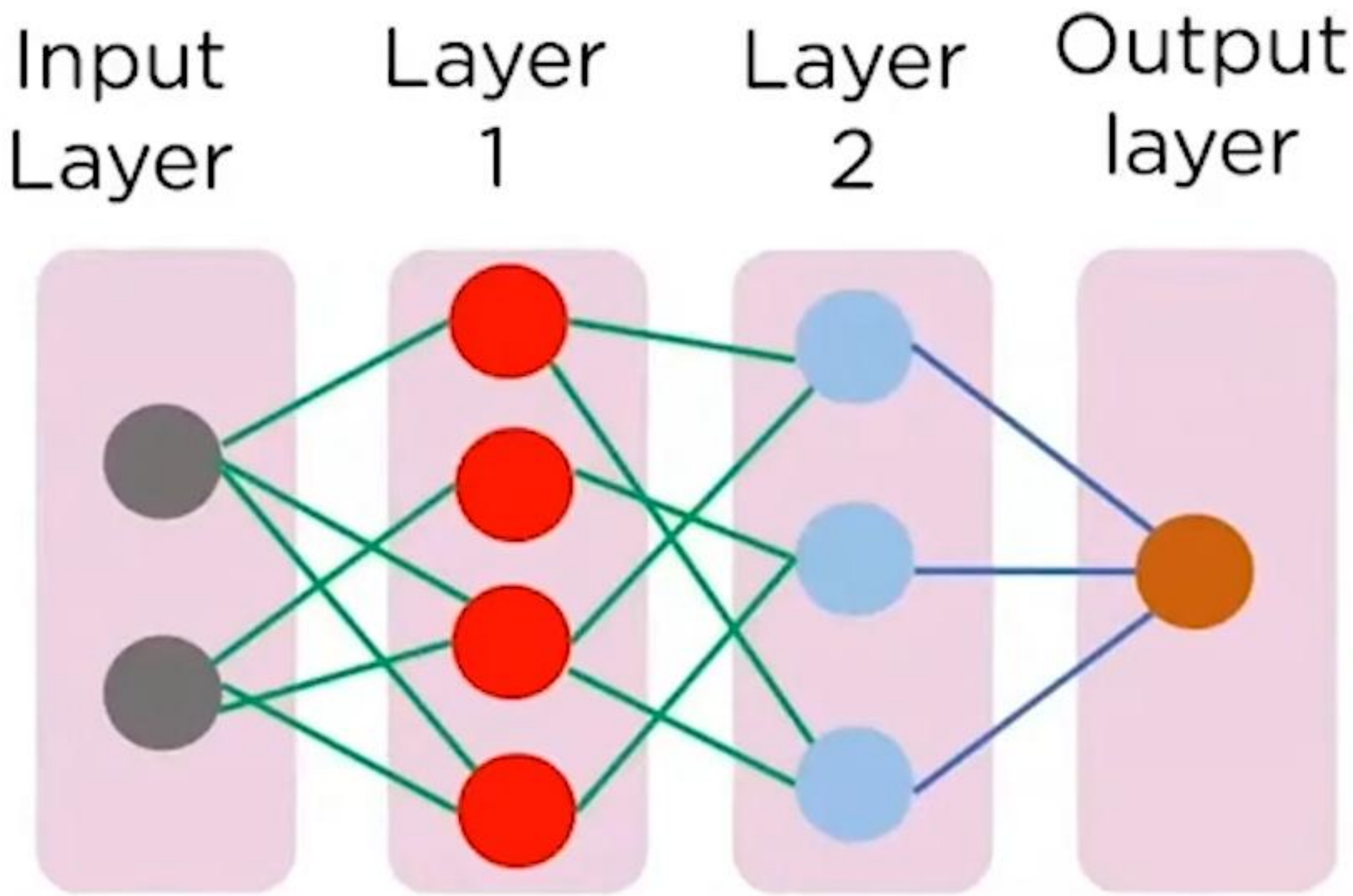
```
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 2)              --> 8 <--
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 3
=================================================================
Total params: 11
Trainable params: 11
Non-trainable params: 0
_________________________________________________________________
```

Keras Model

1 Sequential Model

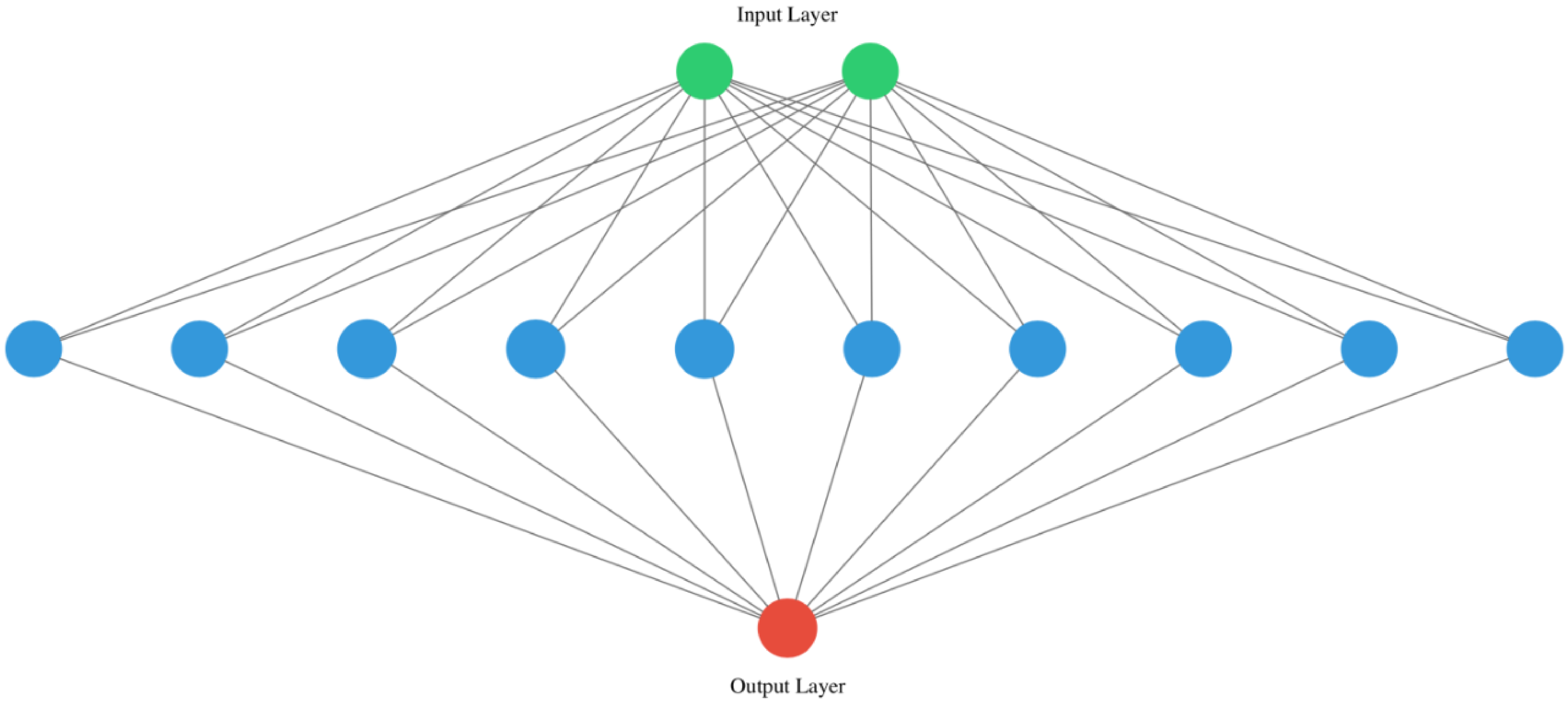
- Sequential Model is a linear stack of layers where the previous layer leads into the next layer
- Useful for simple classifier or decoder models

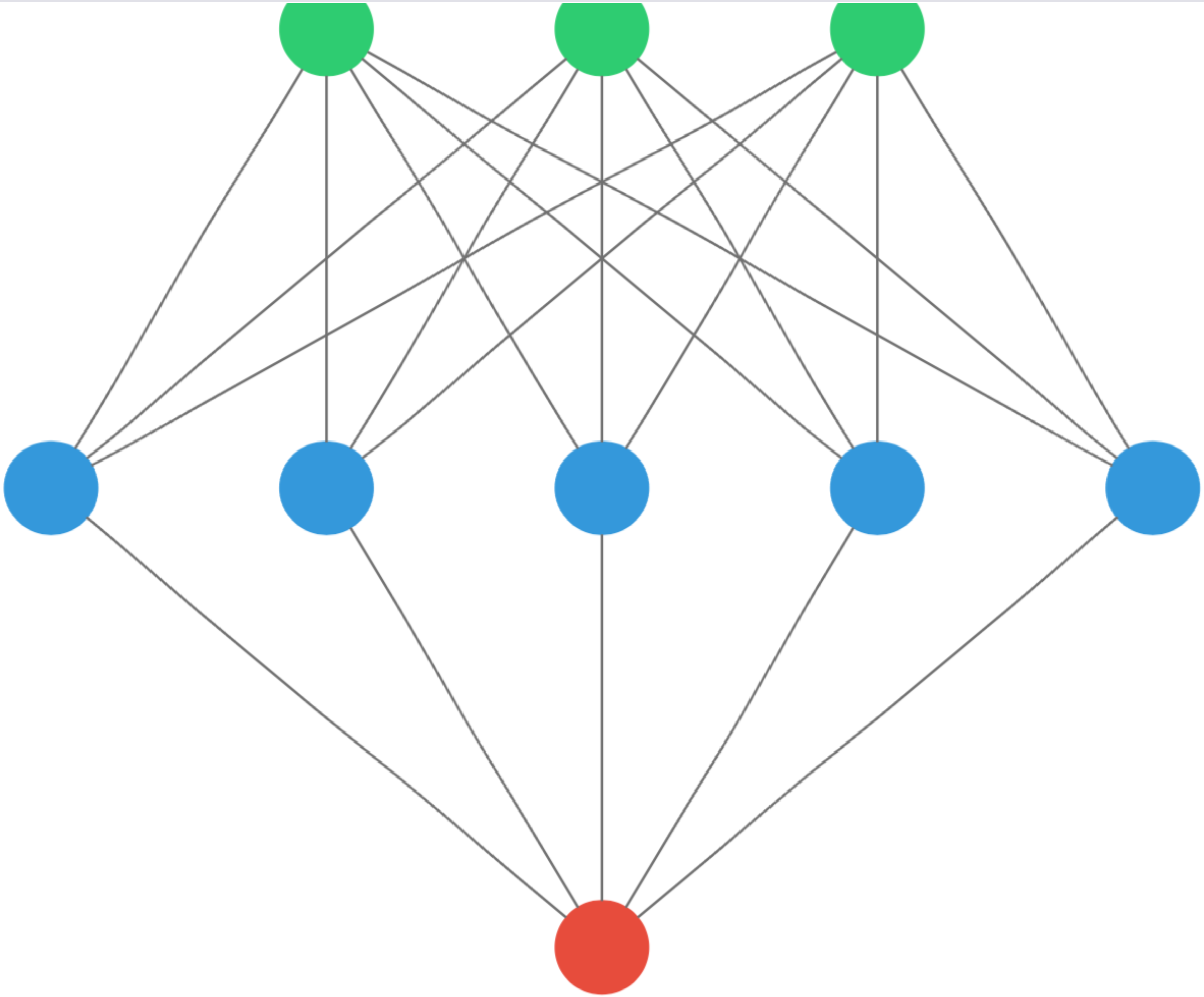
```
model = keras.Sequential( [
layers.Dense(1, activation="relu", name="layer1"),
layers.Dense(2, activation="relu", name="layer2"),
layers.Dense(3, name="layer3"), ] )
# Call model on a test input x = tf.ones((3, 3)) y =
model(x)
```



Let's code!

INTRODUCTION TO DEEP LEARNING WITH KERAS


Input Layer
Output Layer



Output Layer

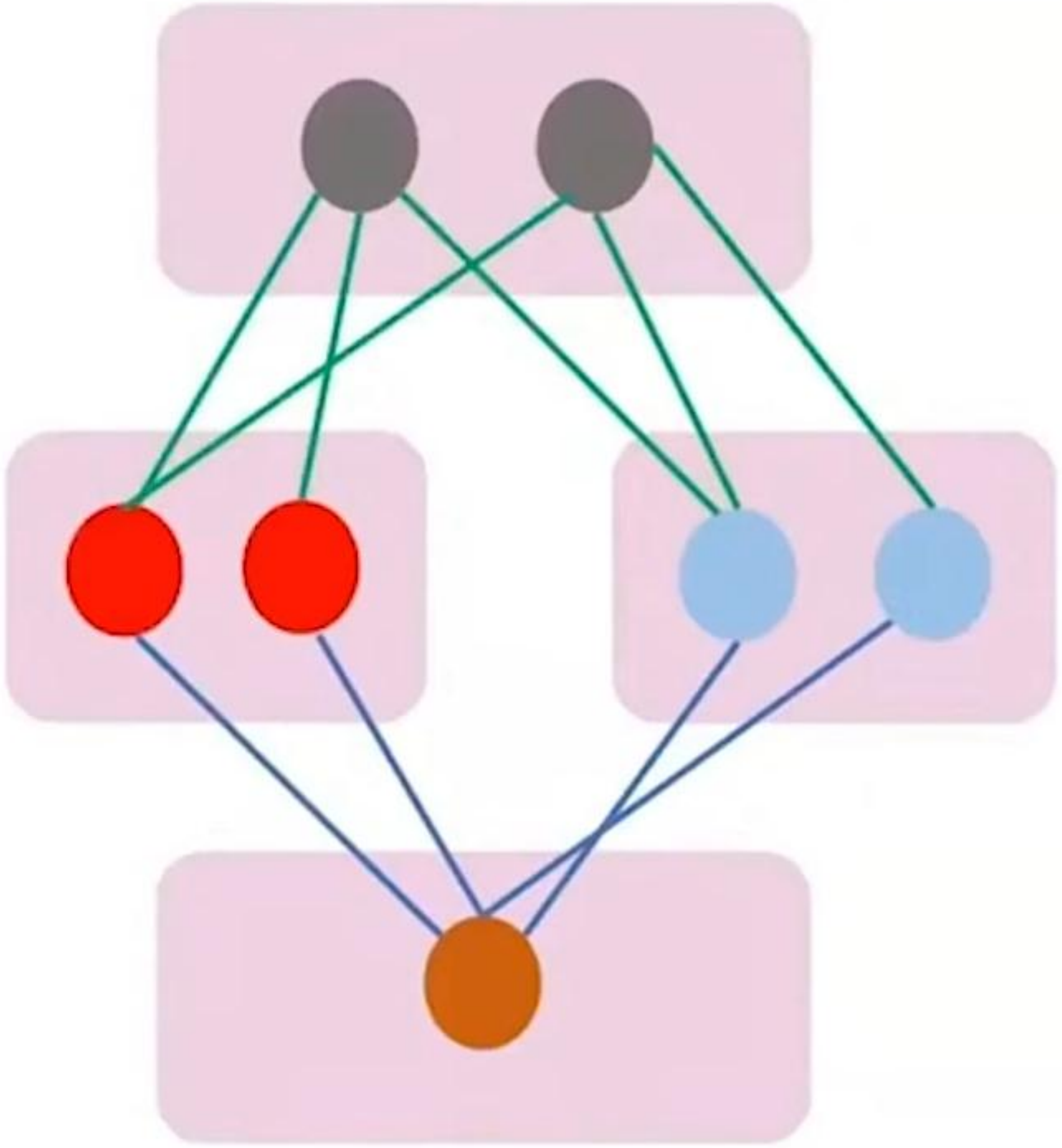
Keras Model

2 Functional Model

- Multi input and multi output model

- Complex model which forks into two or more branches

```
img_inputs = keras.Input(shape=(32, 32, 3))
dense = layers.Dense(64,
activation="relu") x = dense(inputs)
x = layers.Dense(64, activation="relu")
(x) outputs = layers.Dense(10)(x)
model = keras.Model(inputs=inputs,
outputs=outputs,  name="mnist_model")
```

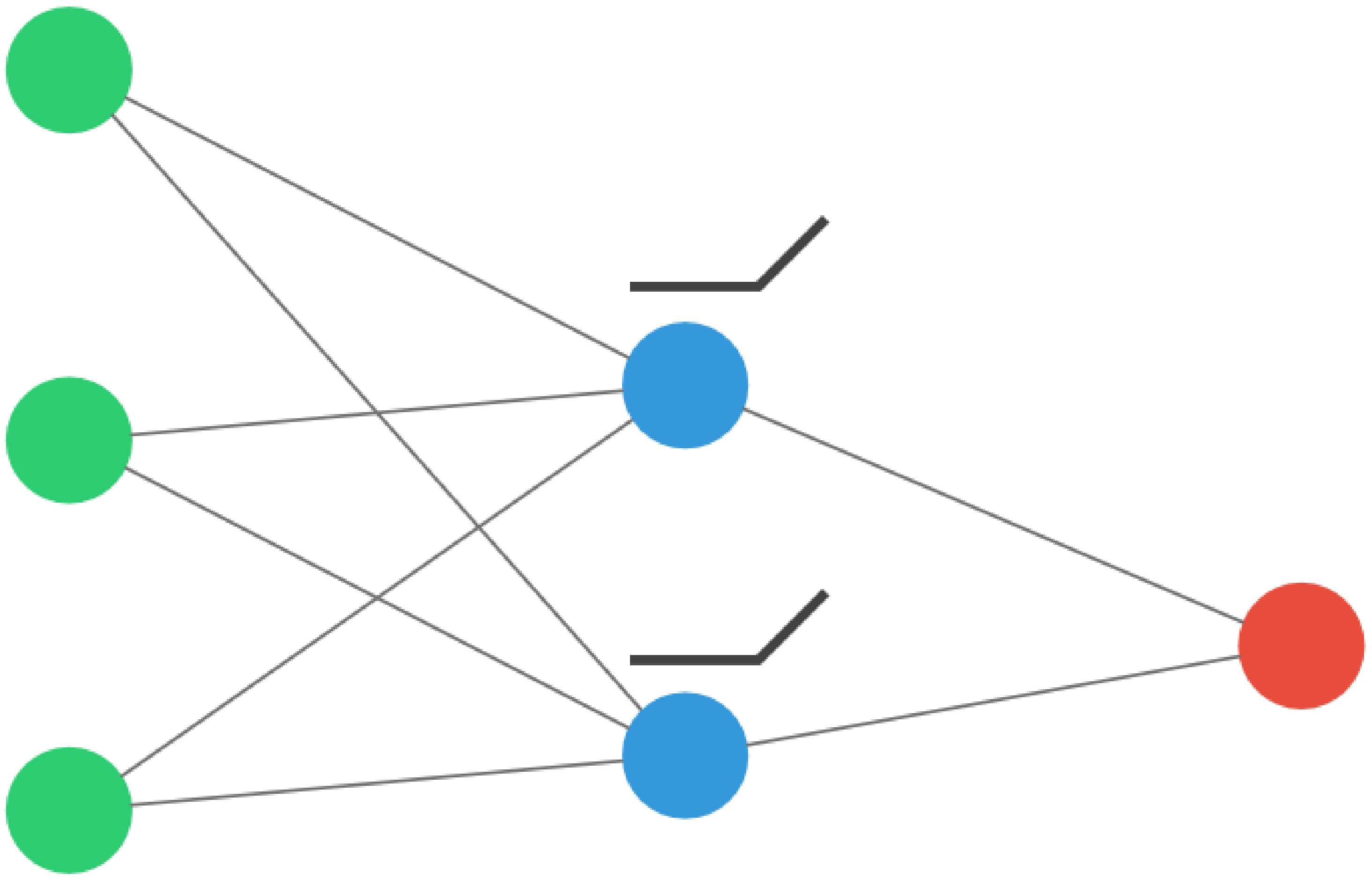


Surviving a meteor strike

INTRODUCTION TO DEEP LEARNING WITH KERAS

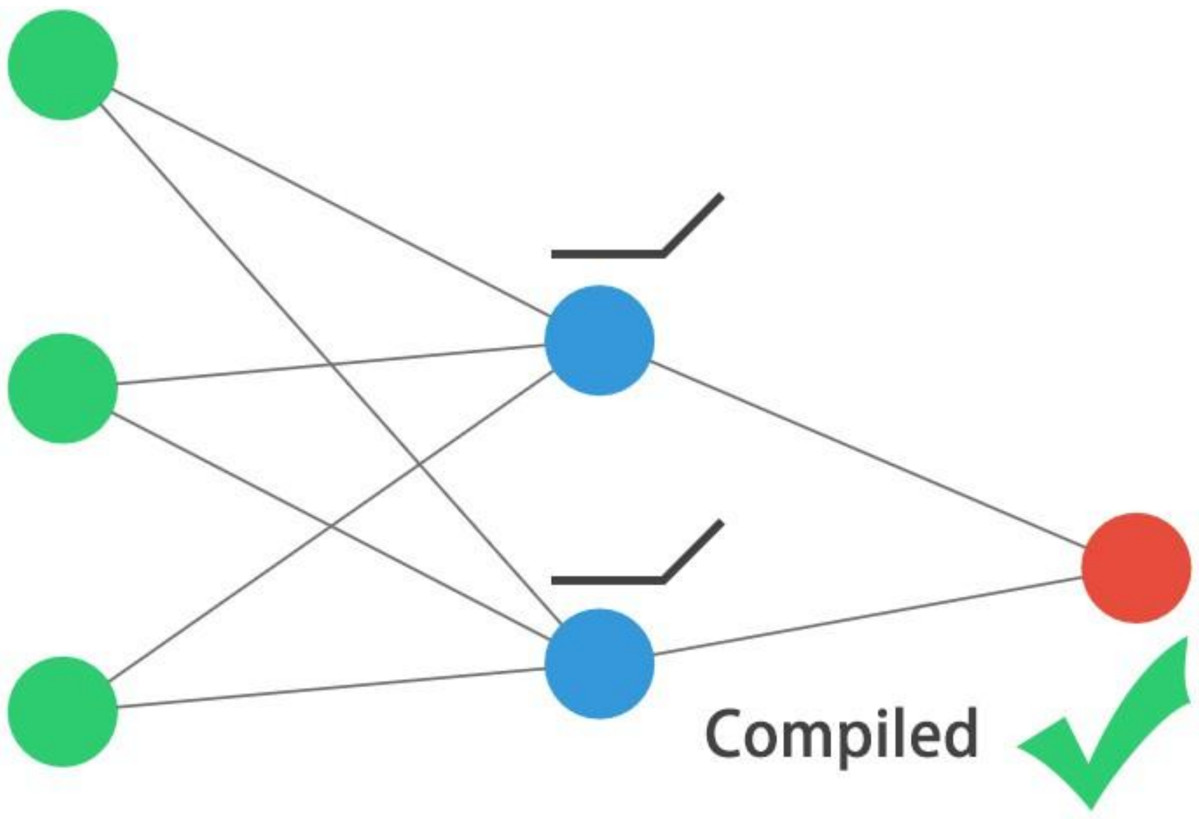
Recap

```python
from tensorflow.keras.models import Sequ
from tensorflow.keras.layers import Dens


# Create a new sequential model
model = Sequential()
# Add and input and dense layer
model.add(Dense(2, input_shape=(3,),
                activation="relu"))


# Add a final 1 neuron layer
model.add(Dense(1))
<
```



Compiling

```
# Compiling your previously built model
model.compile(optimizer="adam", loss="mse")
```



Training

```
# Train your model
model.fit(X_train, y_train, epochs=5)
```

```
Epoch 1/5
1000/1000 [==============================] - 0s 242us/step - loss: 0.4090
Epoch 2/5
1000/1000 [==============================] - 0s 34us/step - loss: 0.3602
Epoch 3/5
1000/1000 [==============================] - 0s 37us/step - loss: 0.3223
Epoch 4/5
1000/1000 [==============================] - 0s 34us/step - loss: 0.2958
Epoch 5/5
1000/1000 [==============================] - 0s 33us/step - loss: 0.2795
```

Predicting

```python
# Predict on new data
preds = model.predict(X_test)

# Look at the predictions
print(preds)
```

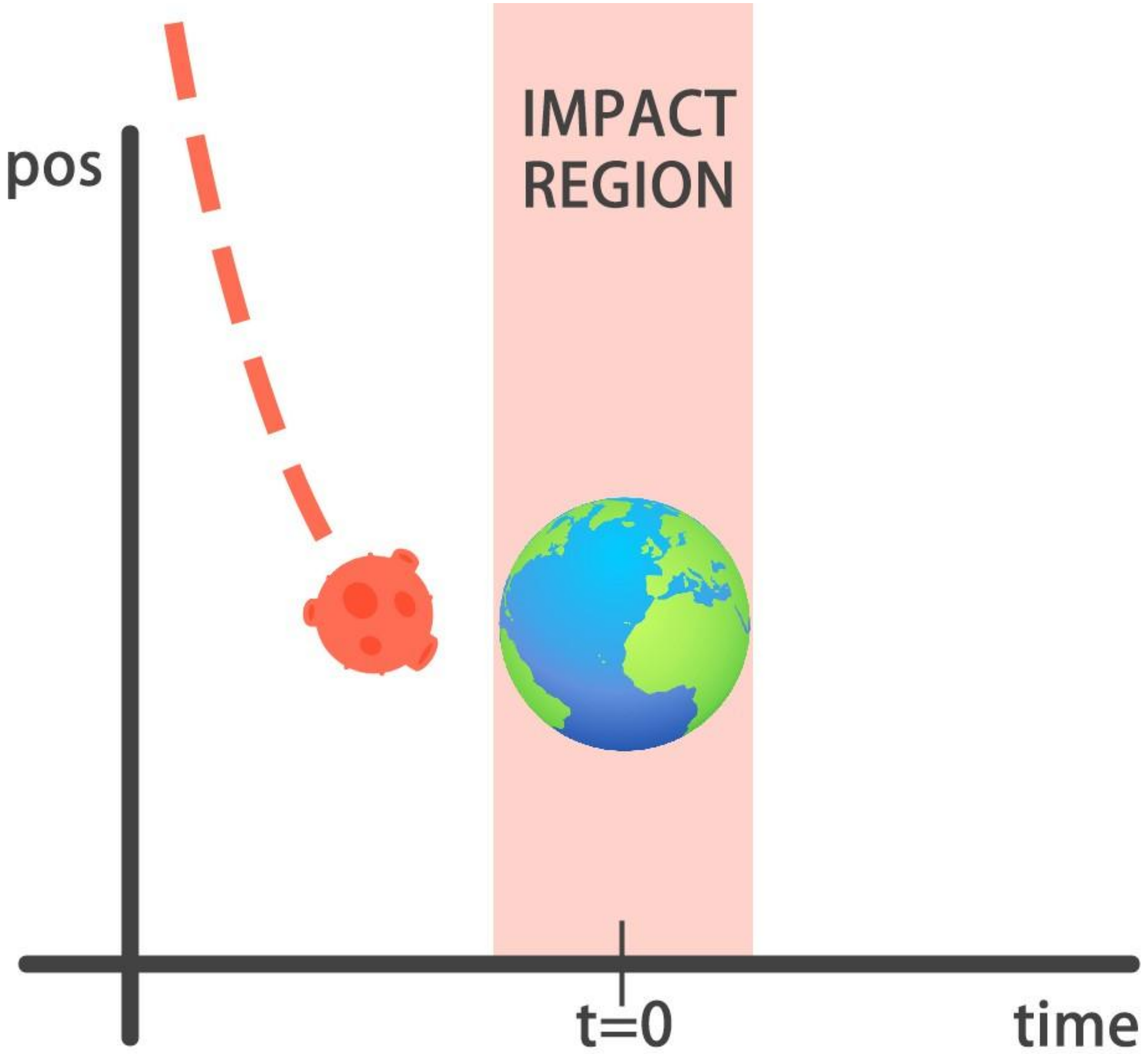
```
array([[0.6131608 ],
       [0.5175948 ],
       [0.60209155],
       ...,
       [0.55633   ],
       [0.5305591 ],
       [0.50682044]])
```

Evaluating

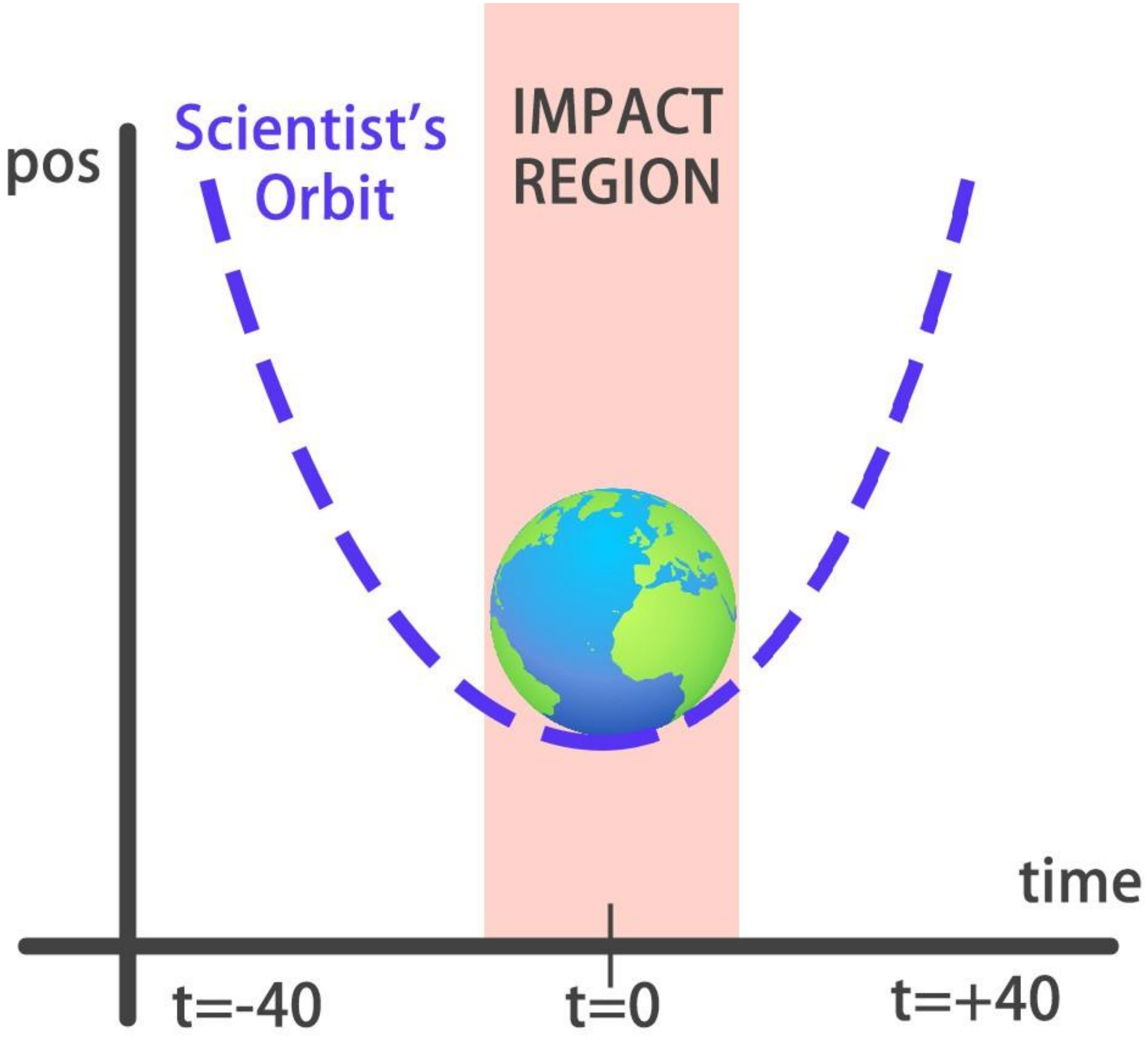
```python
# Evaluate your results
model.evaluate(X_test, y_test)
```

```
1000/1000 [==============================] - 0s 53us/step
0.25
```

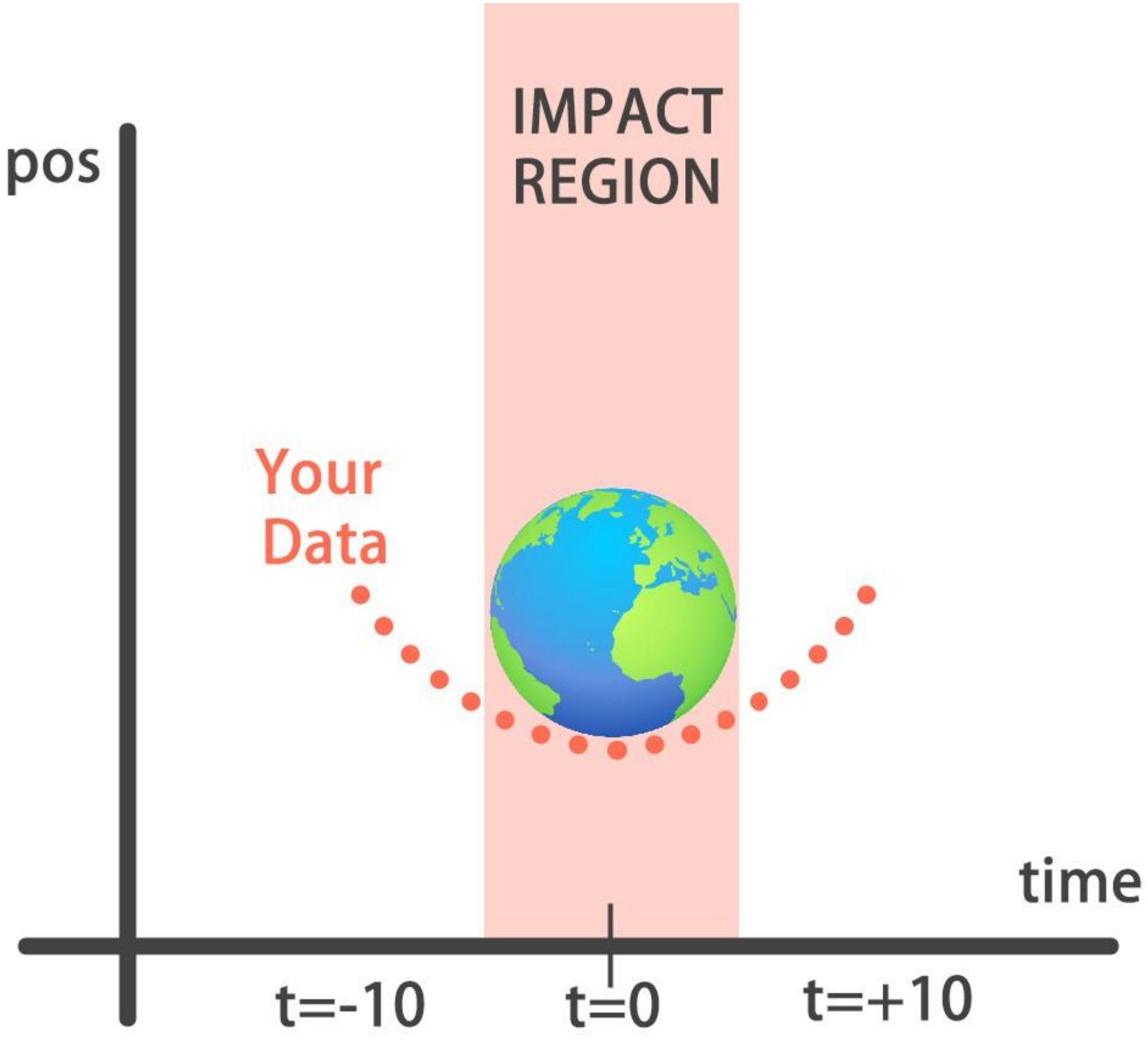
The problem at hand



Scientific prediction



Your task



Let's save the earth!

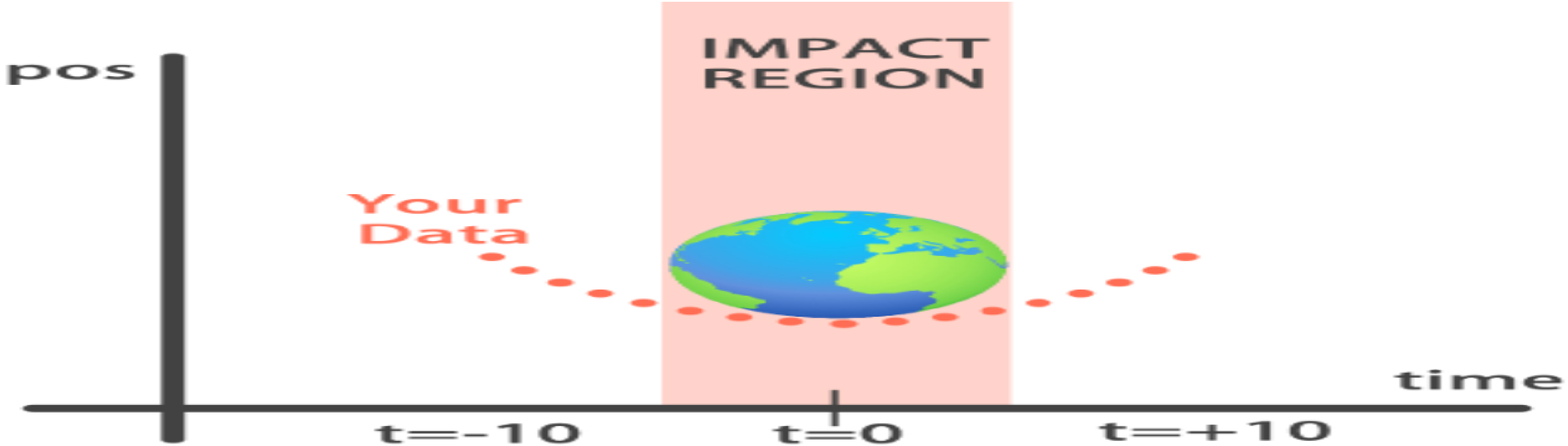
INTRODUCTION TO DEEP LEARNING WITH KERAS

Specifying a model

You will build a simple regression model to predict the orbit of the meteor!

Your training data consist of measurements taken at time steps from -10 minutes before the impact region to +10 minutes after. Each time step can be viewed as an X coordinate in our graph, which has an associated position Y for the meteor orbit at that time step.

Note that you can view this problem as approximating a quadratic function via the use of neural networks.



This data is stored in two numpy arrays: one called `time_steps` , what we call *features*, and another called `y_positions` , with the *labels*. Go on and build your model! It should be able to predict the y positions for the meteor orbit at future time steps.

Keras `Sequential` model and `Dense` layers are available for you to use.

```python
# Instantiate a Sequential model
model = ____

# Add a Dense layer with 50 neurons and an input of 1 neuron
model.add(____(____, input_shape=(____,), activation='relu'))

# Add two Dense layers with 50 neurons and relu activation
model.add(____(____,____=____))
model.____

# End your model with a Dense layer and no activation
model.____
```

```python
# Instantiate a Sequential model
model = Sequential()

# Add a Dense layer with 50 neurons and an input of 1 neuron
model.add(Dense(50, input_shape=(1,), activation='relu'))

# Add two Dense layers with 50 neurons and relu activation
model.add(Dense(50,activation='relu'))
model.add(Dense(50,activation='relu'))


# End your model with a Dense layer and no activation
model.add(Dense(1))
```

```python
# Compile your model
model.compile(optimizer ='adam', loss = 'mse')

print("Training started..., this can take a while:")

# Fit your model on your data for 30 epochs
model.fit(time_steps,y_positions, epochs = 30)

# Evaluate your model
print("Final loss value:",model.evaluate(time_steps,y_positions))
```

```
# Predict the twenty minutes orbit
twenty_min_orbit = model.predict(np.arange(-10, 11))

# Plot the twenty minute orbit
plot_orbit(twenty_min_orbit)
```