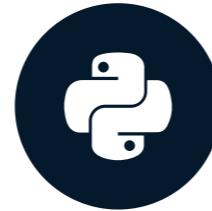


# Binary classification

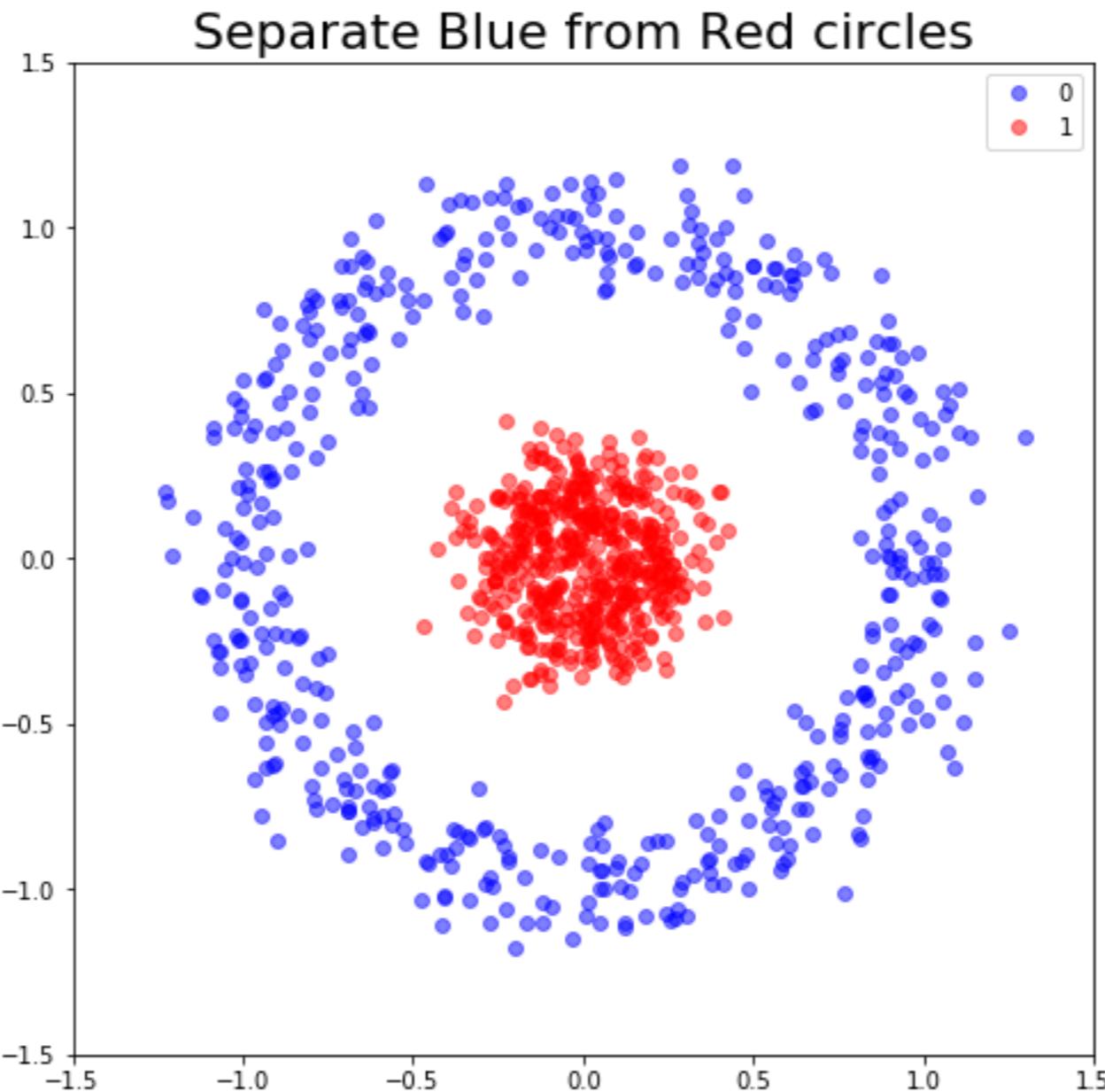
INTRODUCTION TO DEEP LEARNING WITH KERAS



**Miguel Esteban**

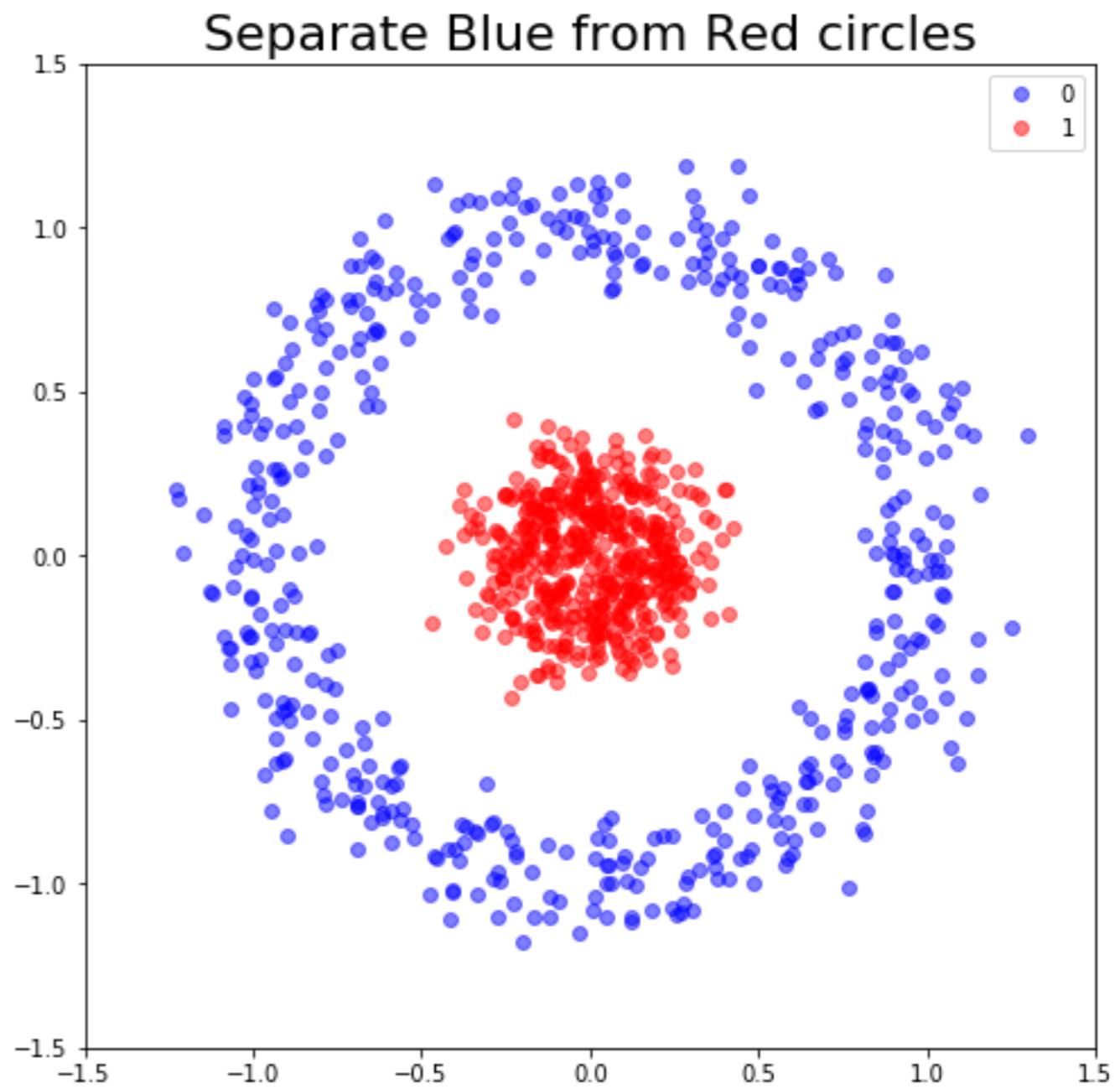
Data Scientist & Founder

# When to use binary classification?



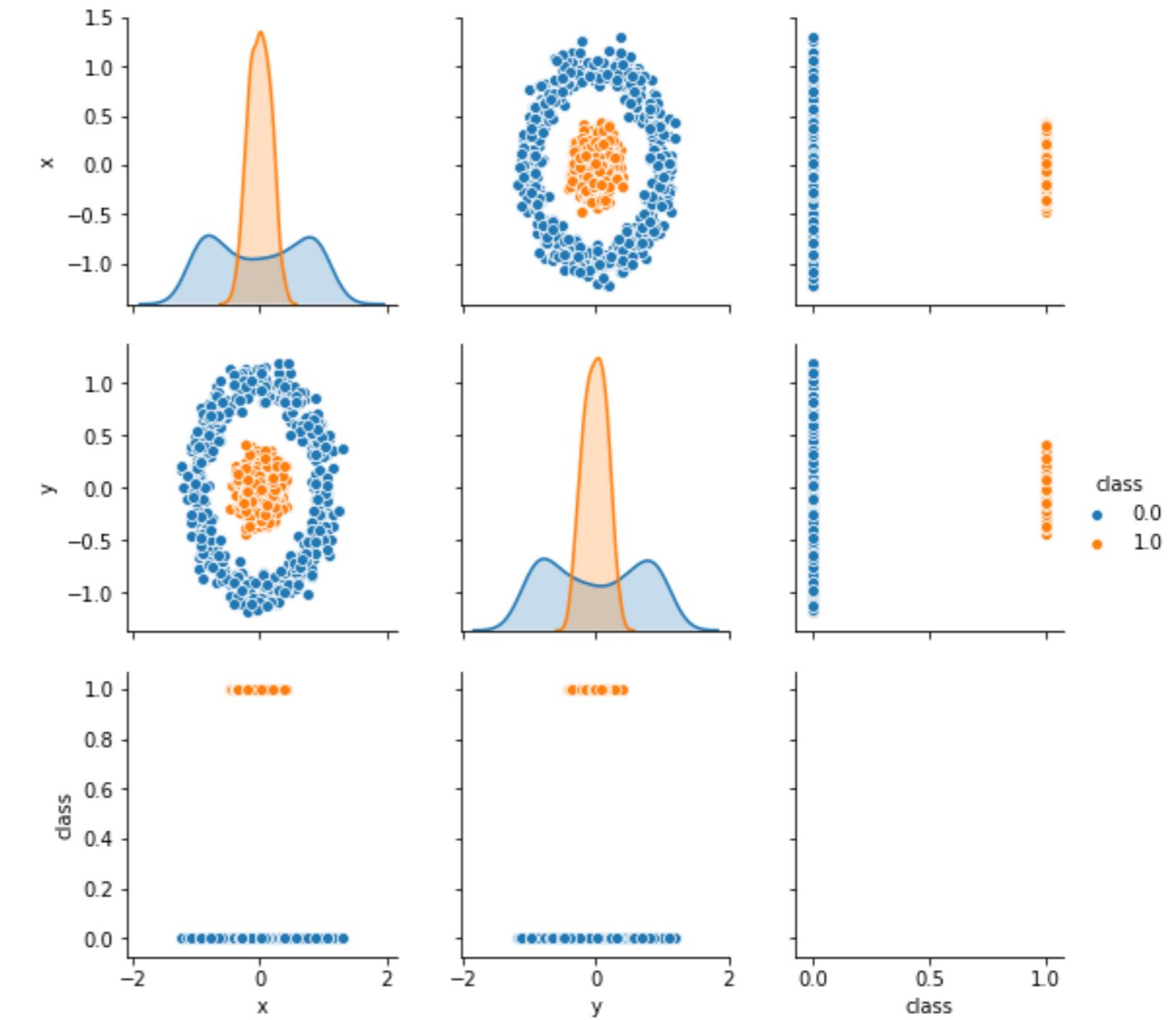
# Our dataset

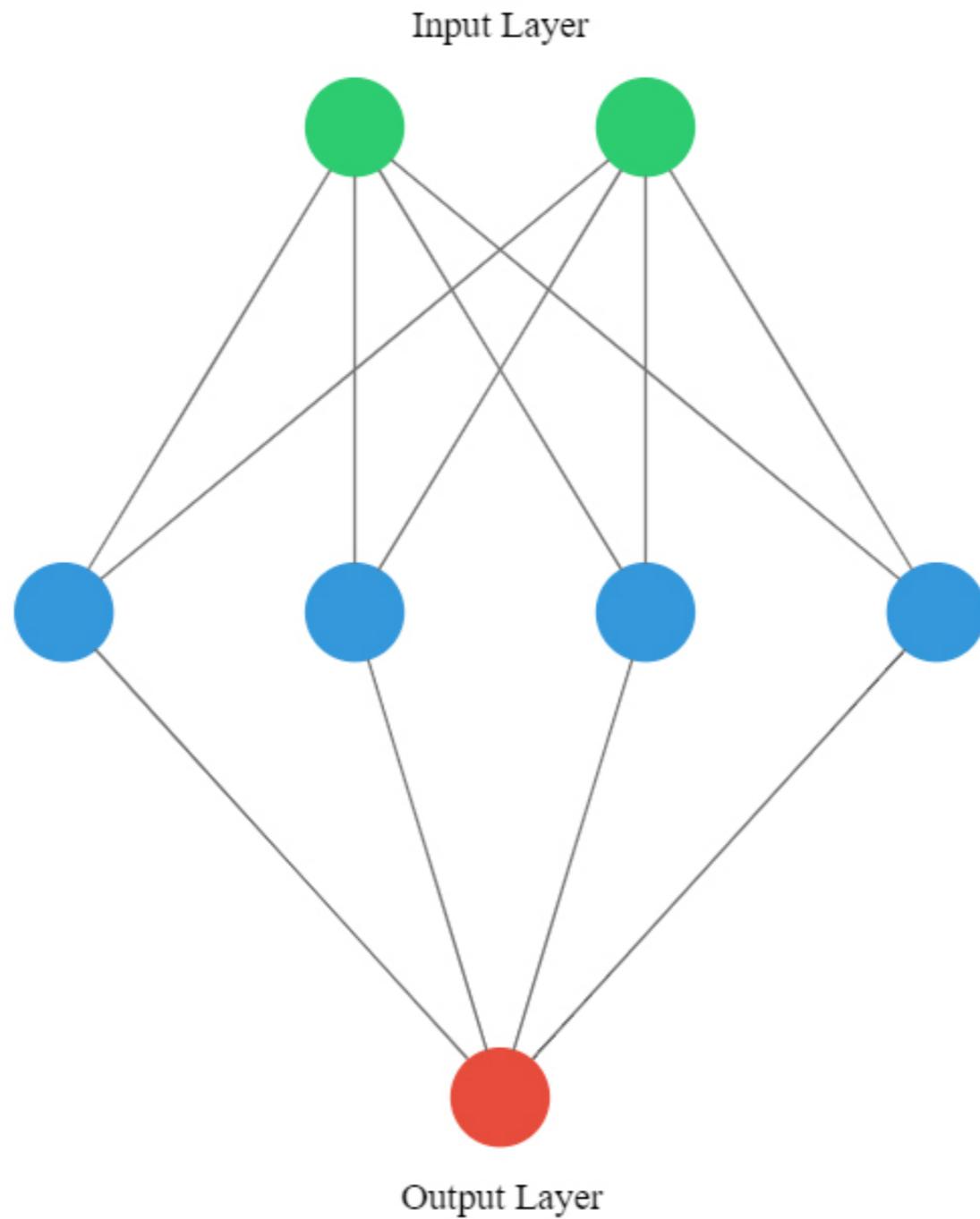
coordinates	labels
[0.242, 0.038]	1
[0.044, -0.057]	1
[-0.787, -0.076]	0

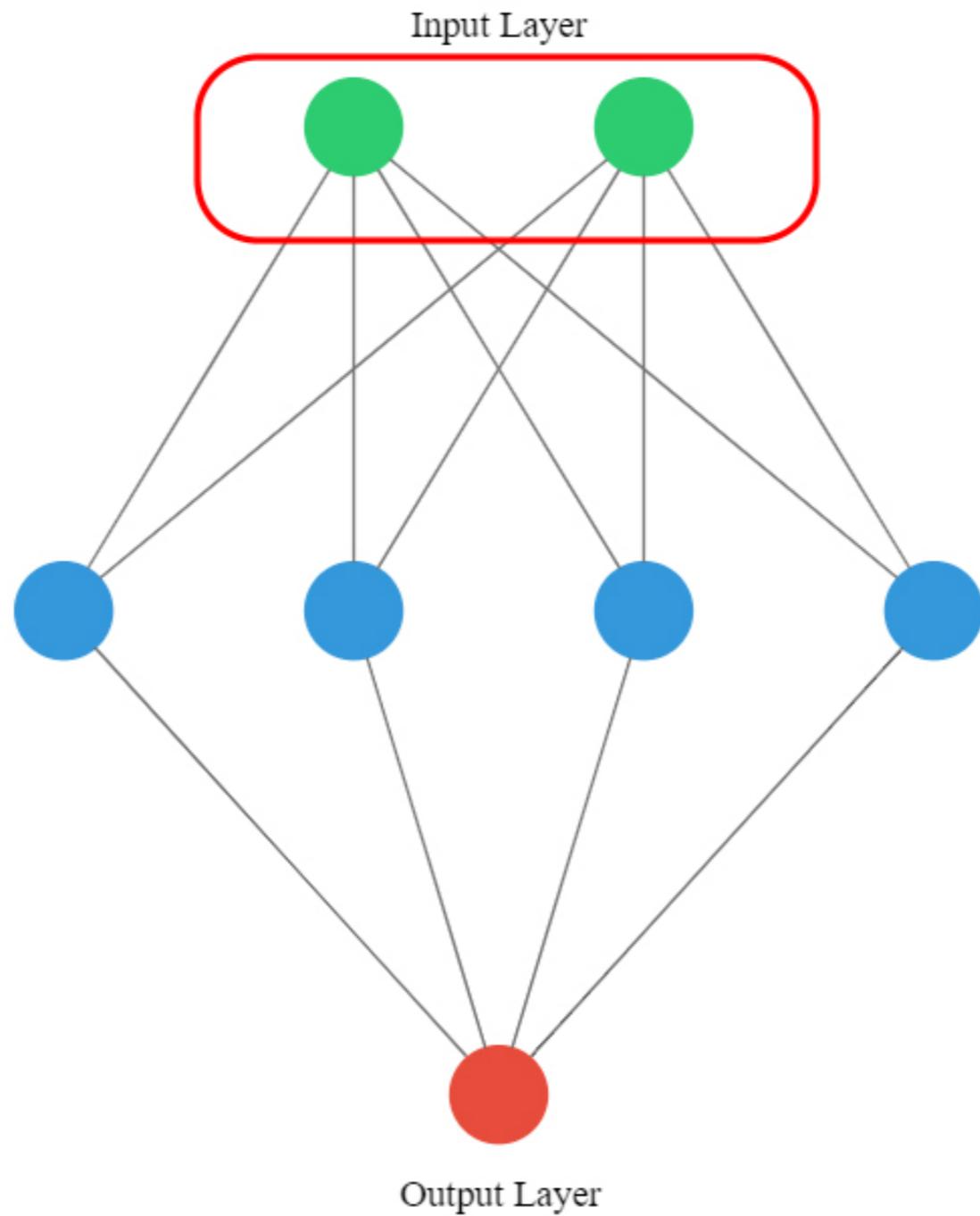


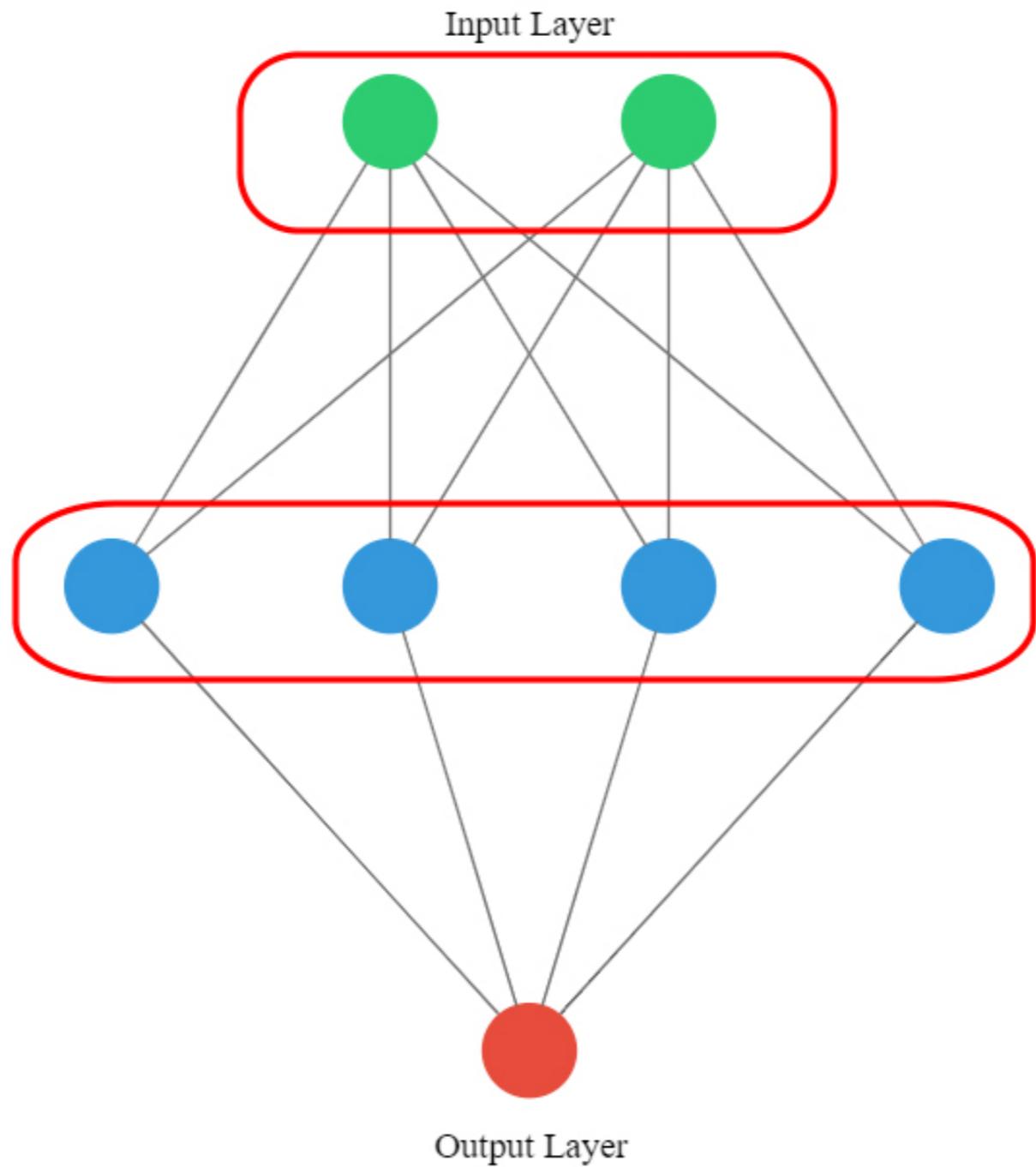
# Pairplots

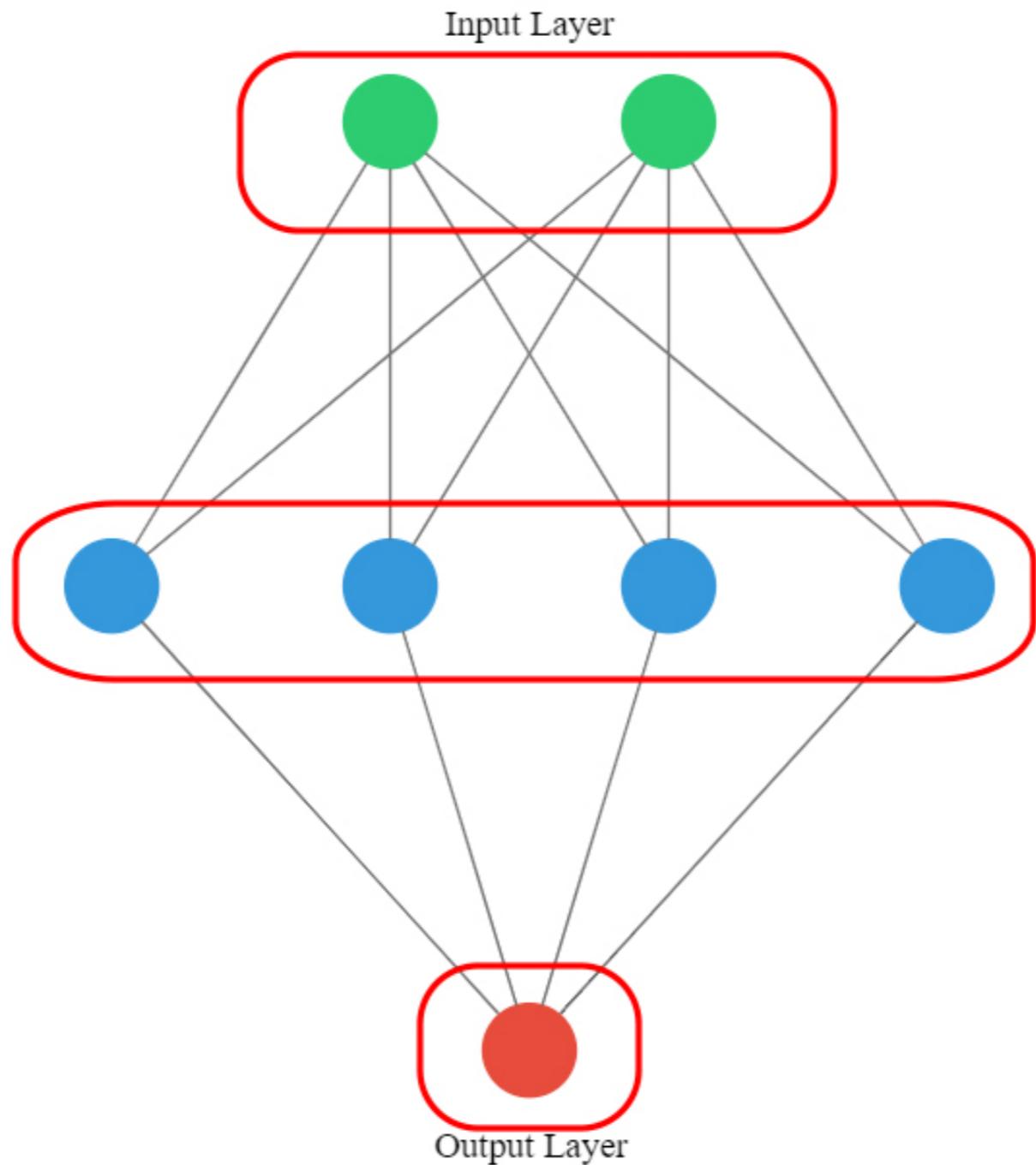
```
import seaborn as sns  
  
# Plot a pairplot  
sns.pairplot(circles, hue="target")
```

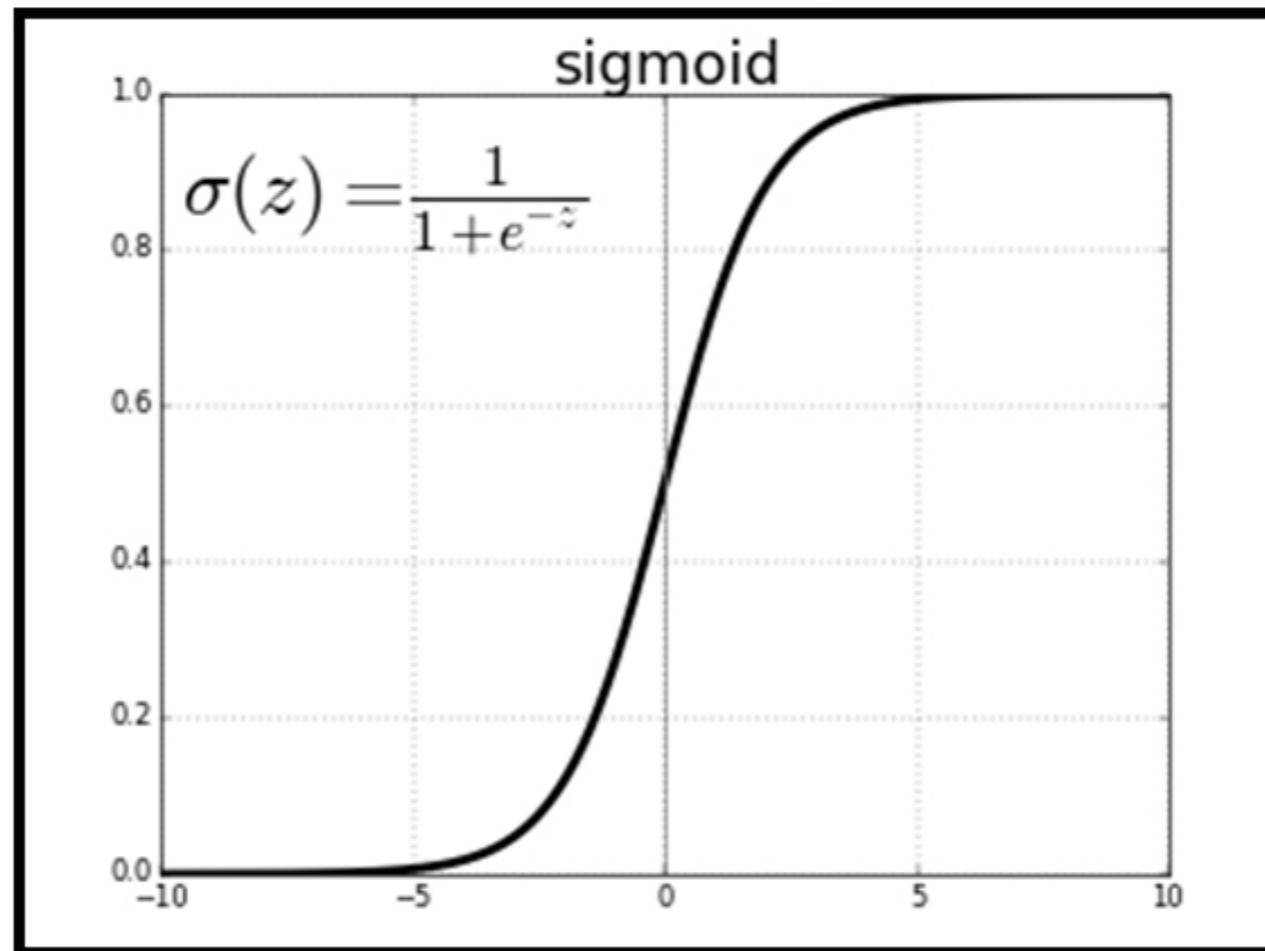












Output Layer

# The sigmoid function

neuron  
output

3 →

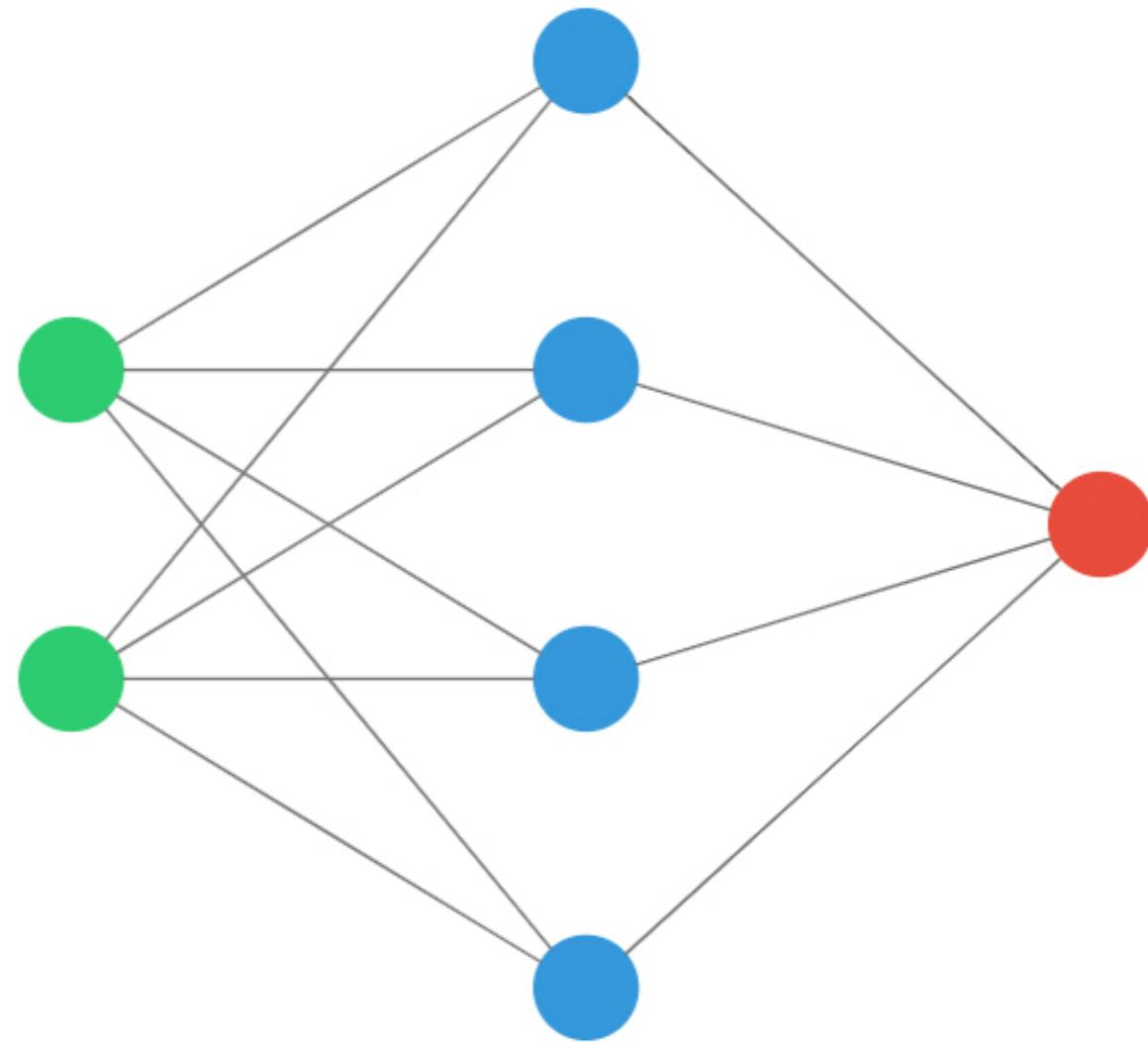
sigmoid

transformed  
output

0.95 → 1

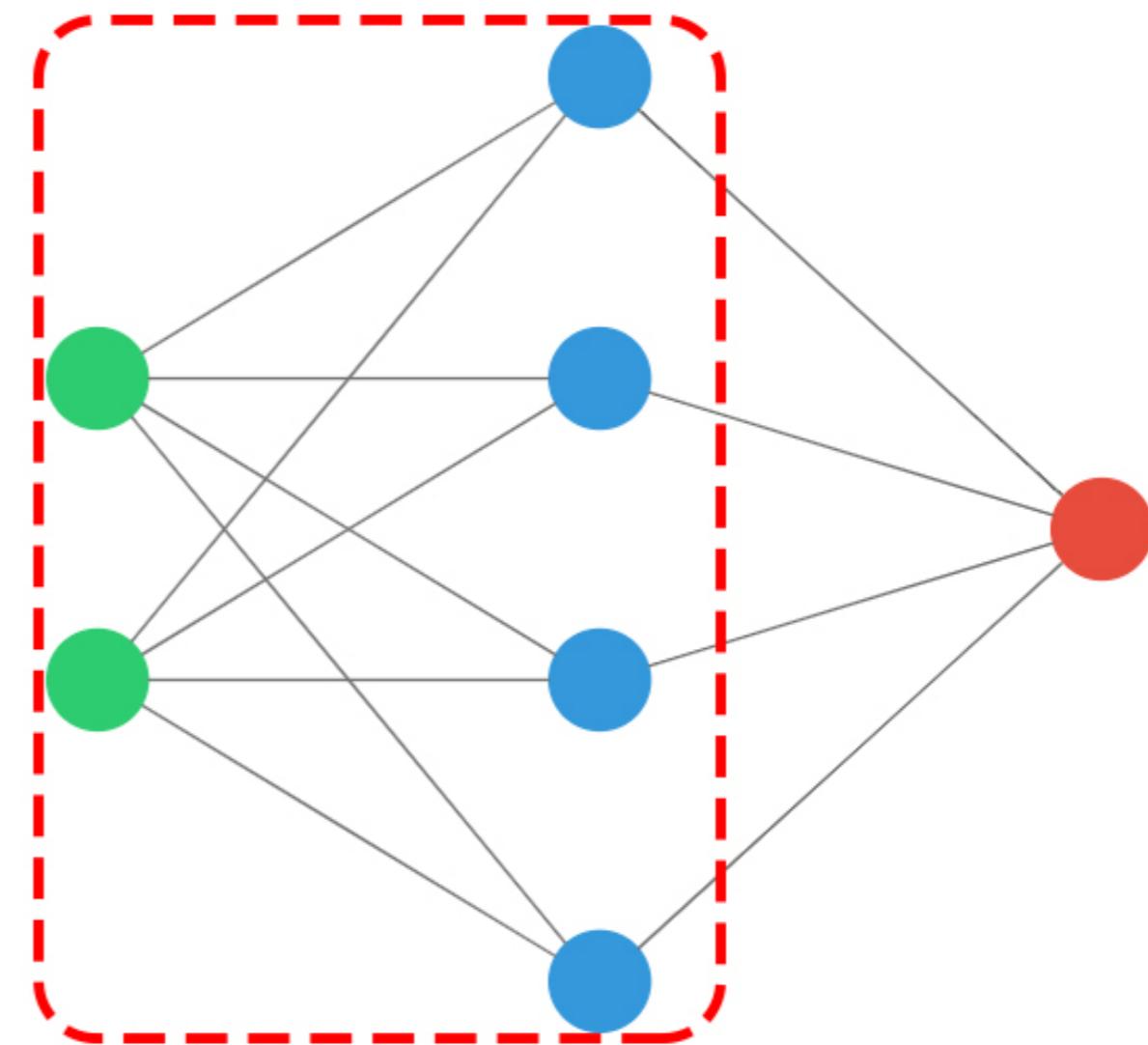
# Let's build it

```
from keras.models import Sequential  
from keras.layers import Dense  
# Instantiate a sequential model  
model = Sequential()
```



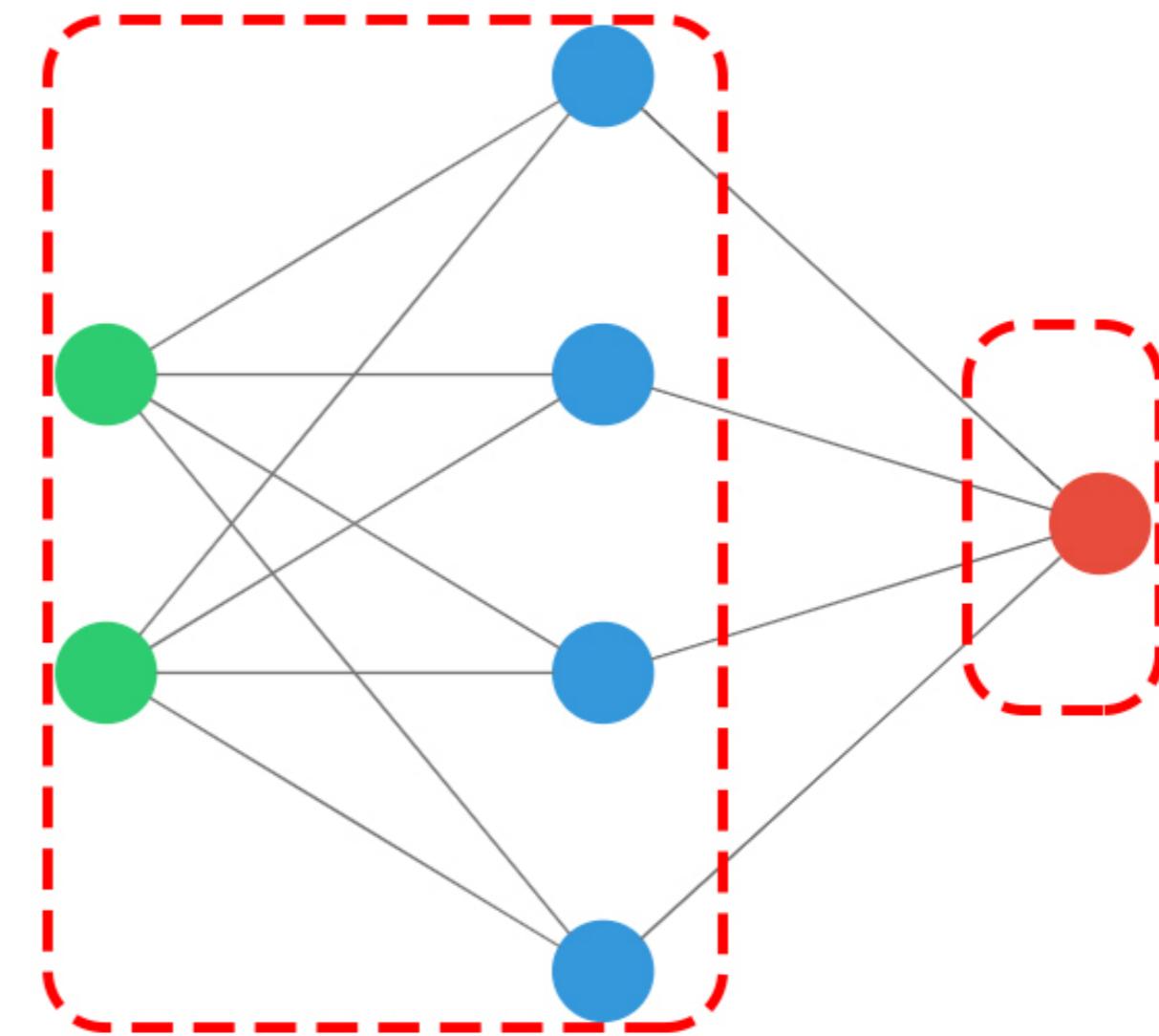
# Let's build it

```
from keras.models import Sequential  
from keras.layers import Dense  
  
# Instantiate a sequential model  
model = Sequential()  
  
# Add input and hidden layer  
model.add(Dense(4, input_shape=(2,),  
               activation='tanh'))
```



# Let's build it

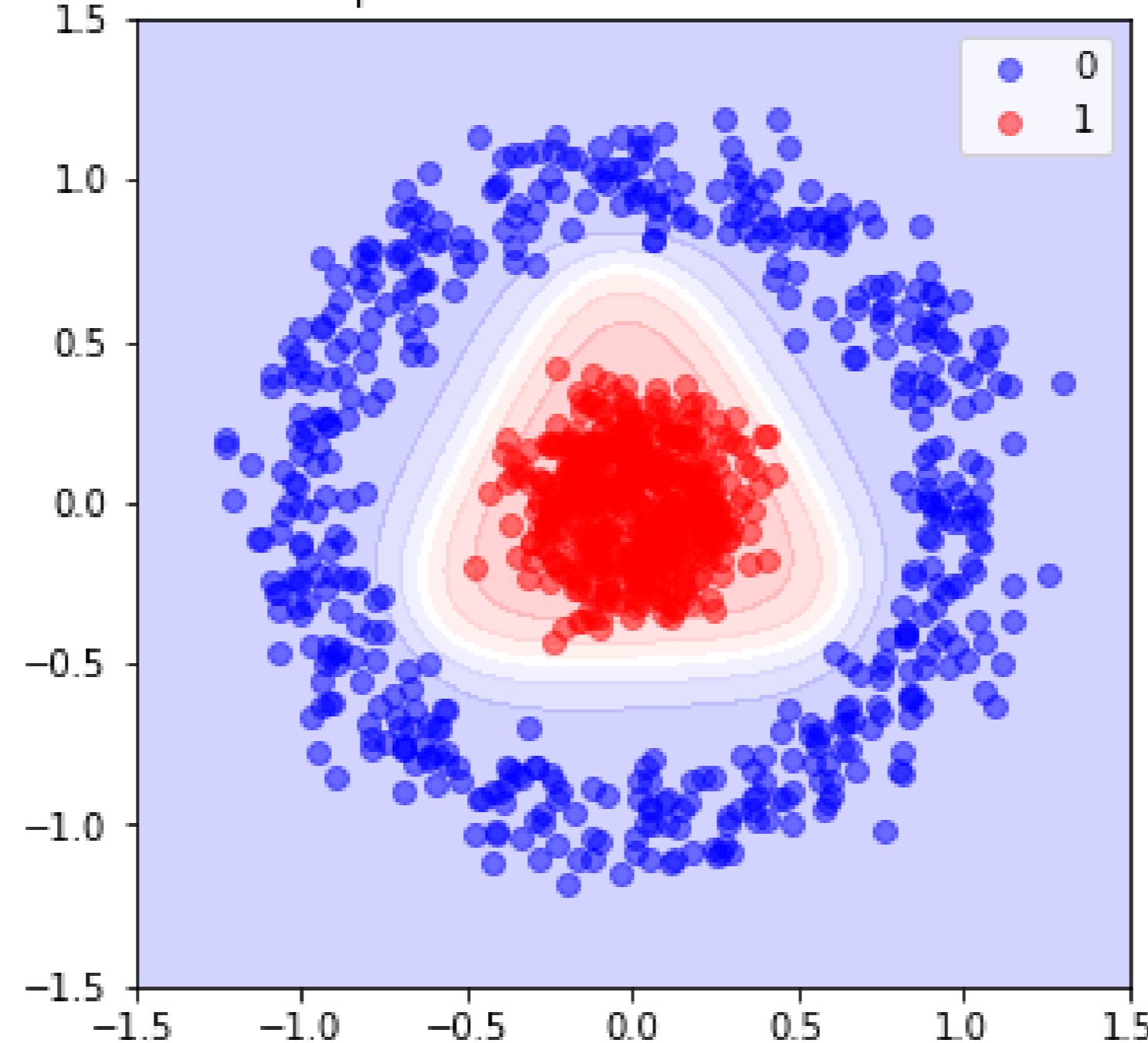
```
from keras.models import Sequential  
from keras.layers import Dense  
  
# Instantiate a sequential model  
model = Sequential()  
  
# Add input and hidden layer  
model.add(Dense(4, input_shape=(2,),  
               activation='tanh'))  
  
# Add output layer, use sigmoid  
model.add(Dense(1,  
               activation='sigmoid'))
```



# Compiling, training, predicting

```
# Compile model  
model.compile(optimizer='sgd',  
              loss='binary_crossentropy')  
  
# Train model  
model.train(coordinates, labels, epochs=20)  
  
# Predict with trained model  
preds = model.predict(coordinates)
```

## Separate Blue from Red circles

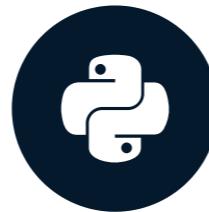


# **Let's practice!**

**INTRODUCTION TO DEEP LEARNING WITH KERAS**

# Multi-class classification

INTRODUCTION TO DEEP LEARNING WITH KERAS



**Miguel Esteban**

Data Scientist & Founder

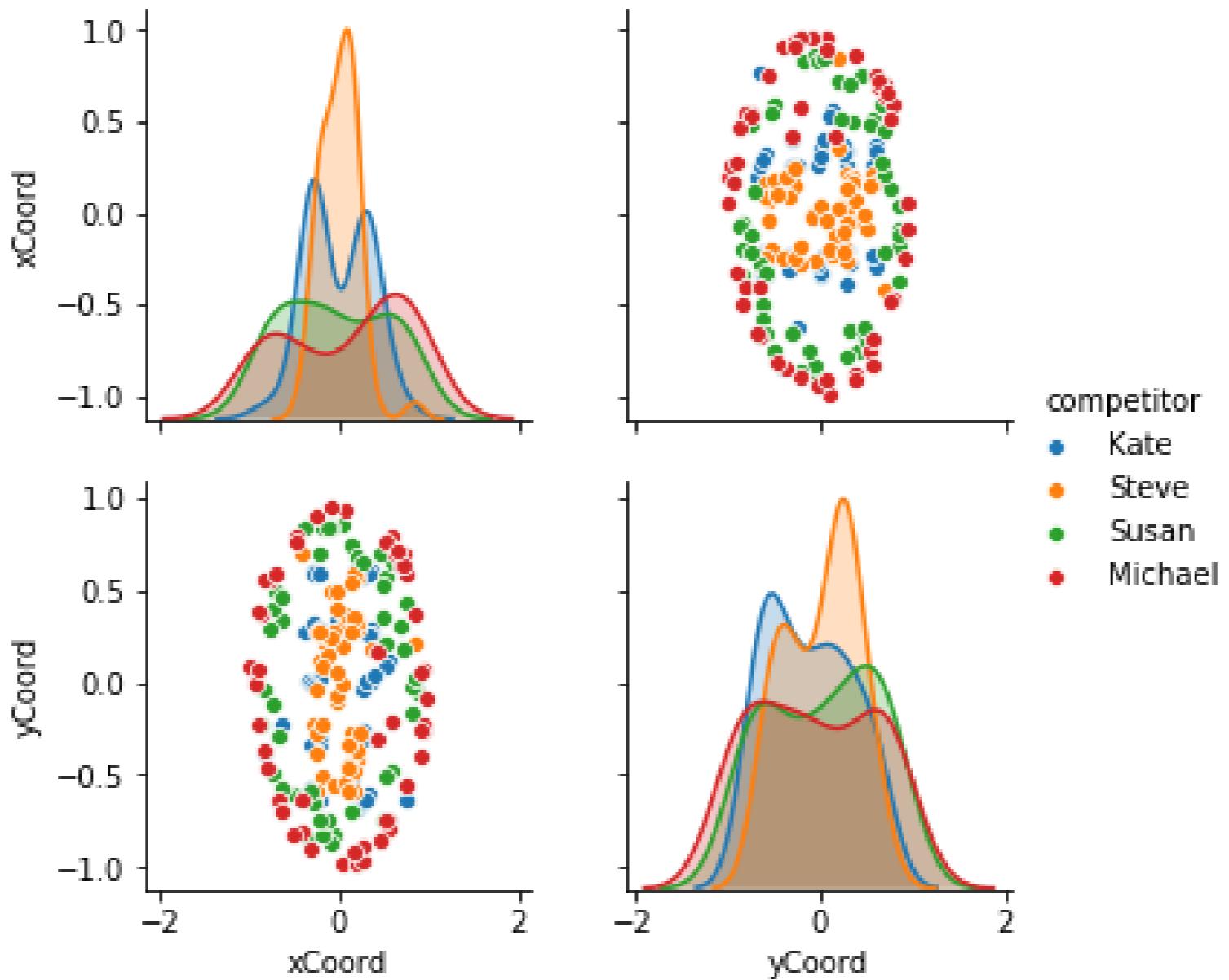
# Throwing darts



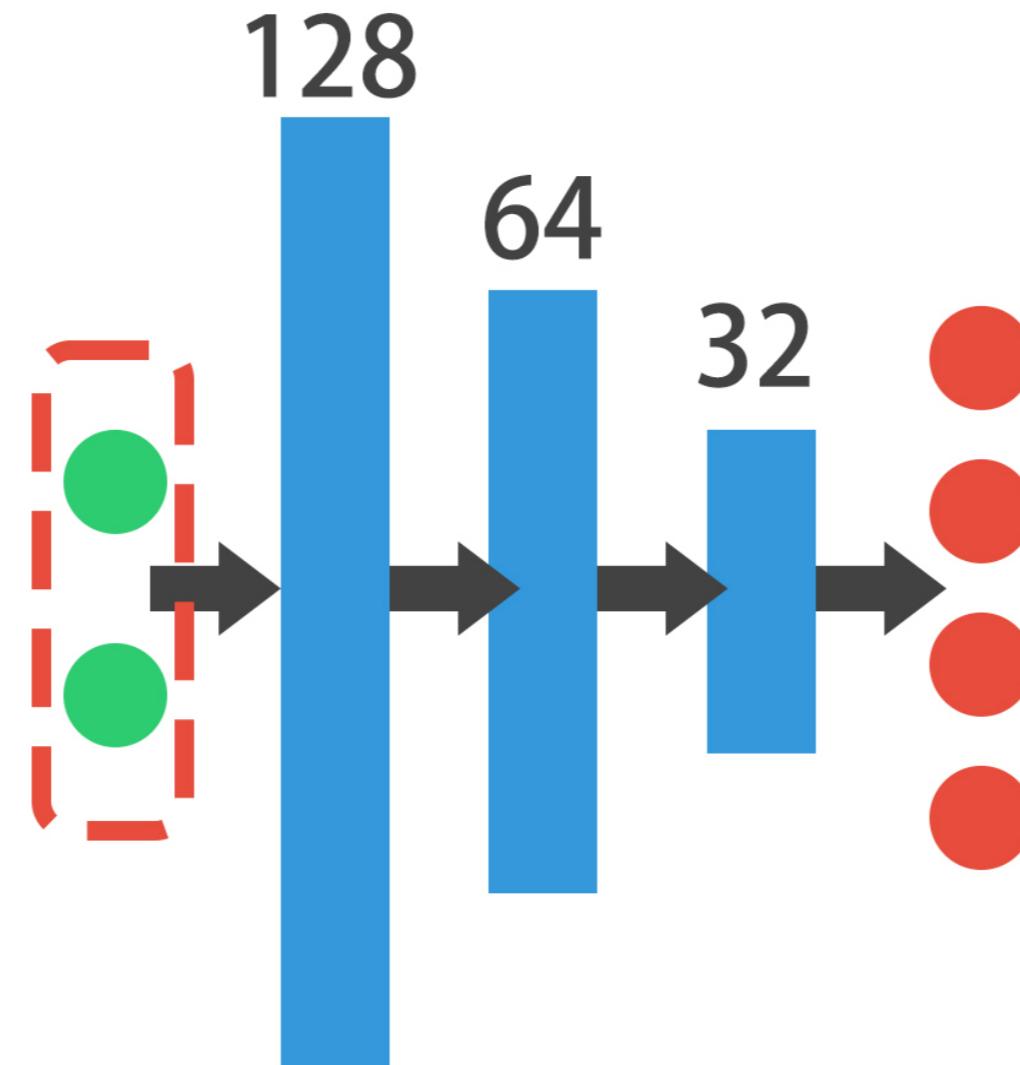
# The dataset

xCoord	yCoord	competitor
-0.037673	0.057402	Steve
-0.331021	-0.585035	Susan
-0.123567	0.839730	Susan
-0.086160	0.959787	Michael
-0.902632	0.078753	Michael

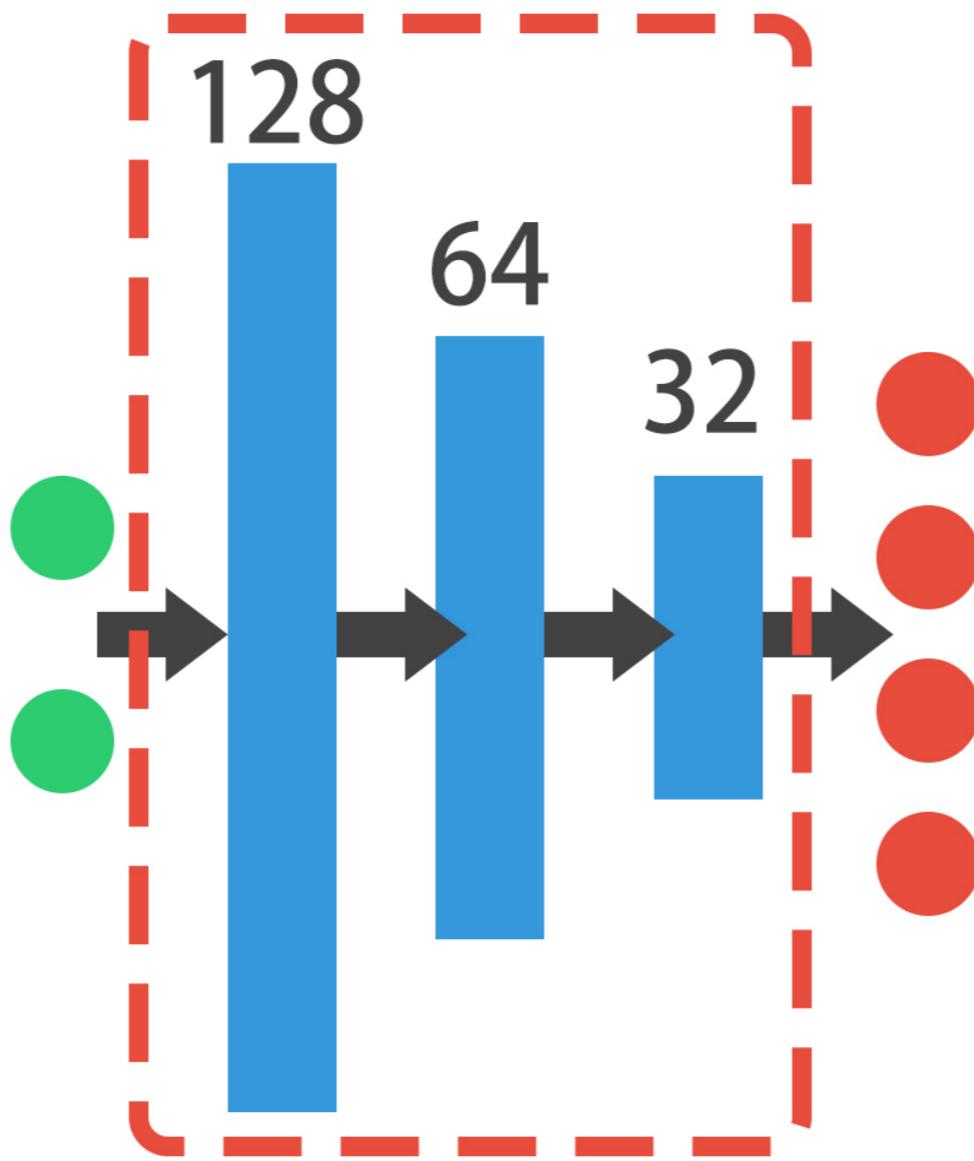
# The dataset



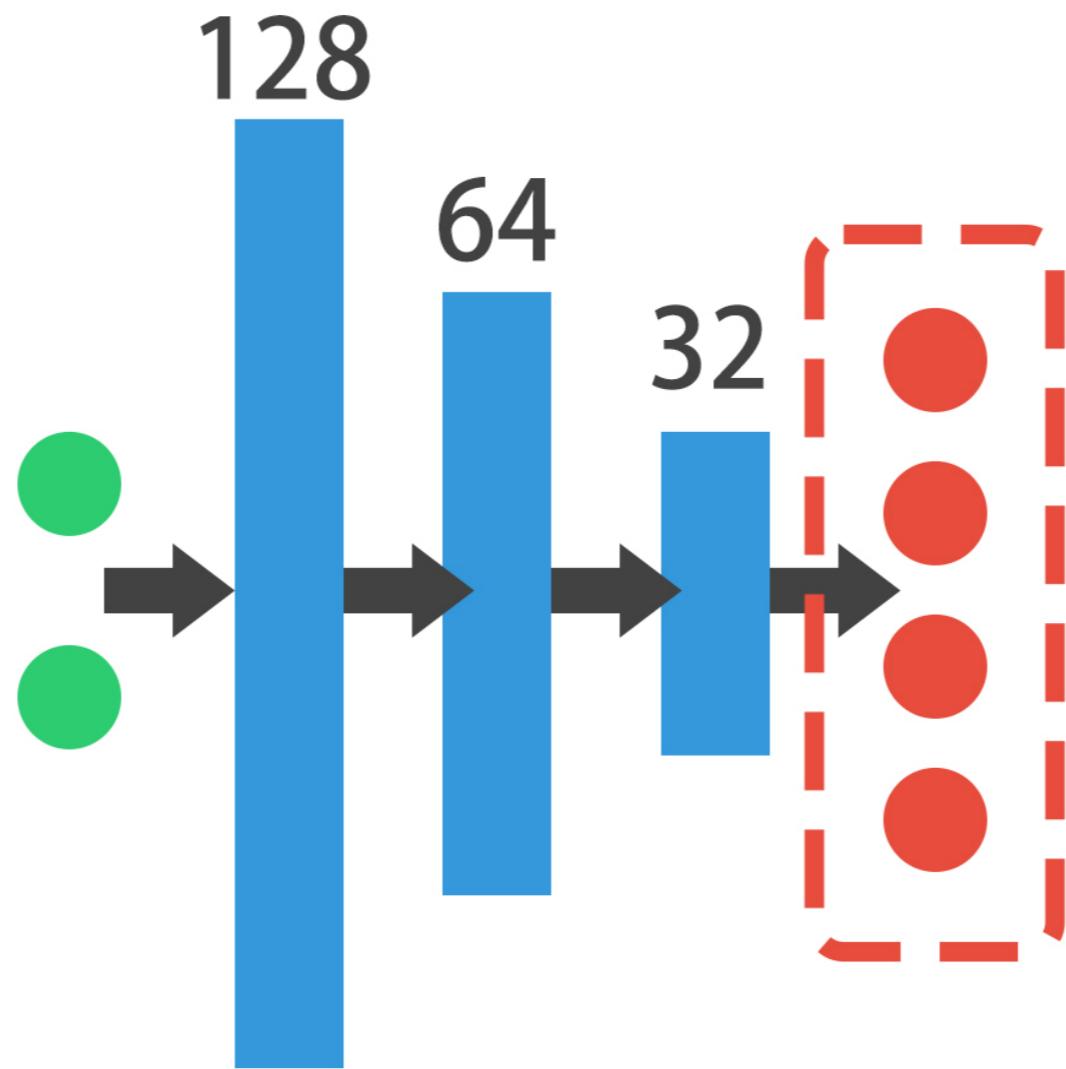
# The architecture



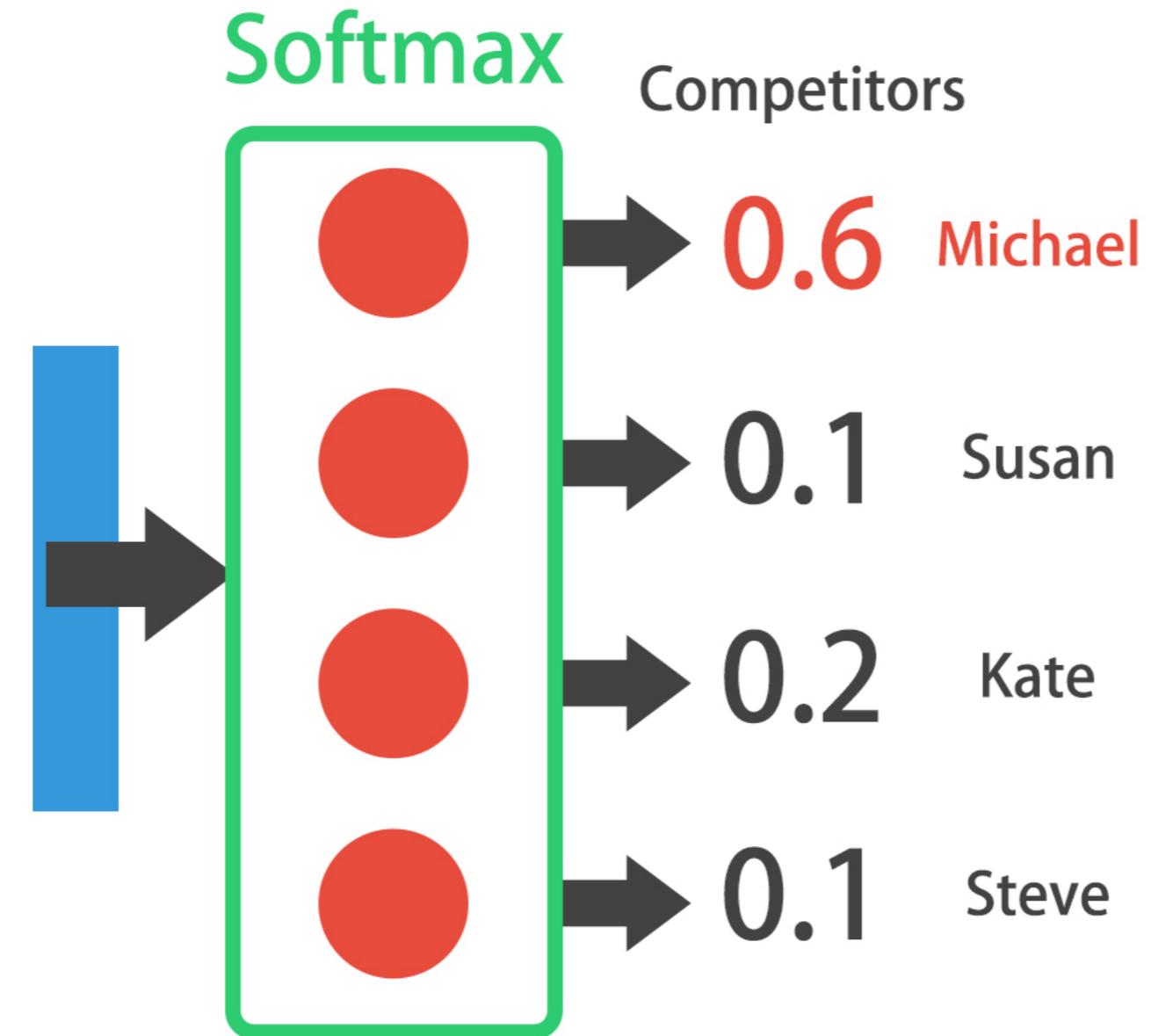
# The architecture



# The architecture



# The output layer

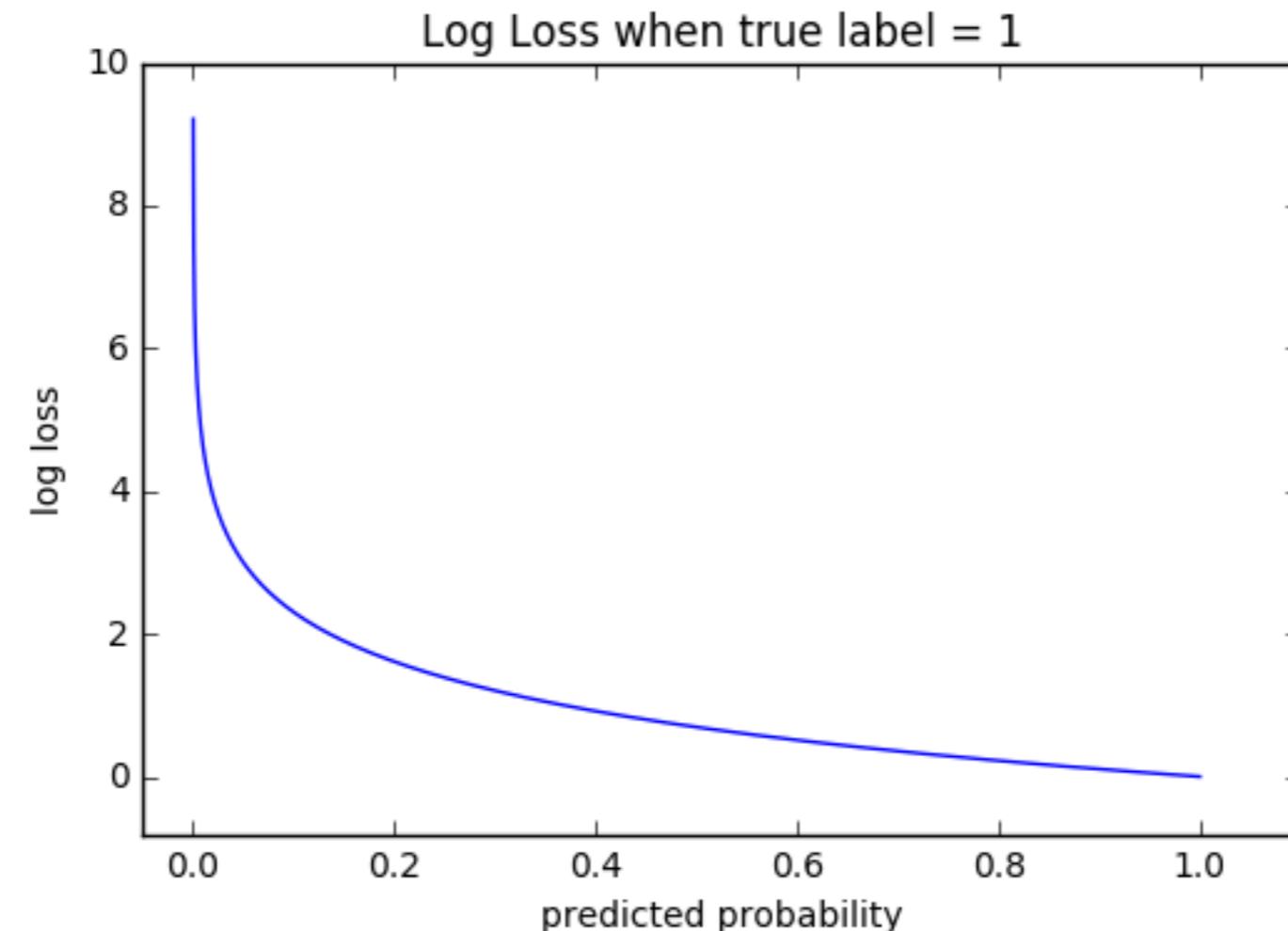


# Multi-class model

```
# Instantiate a sequential model  
# ...  
# Add an input and hidden layer  
# ...  
# Add more hidden layers  
# ...  
# Add your output layer  
model.add(Dense(4, activation='softmax'))
```

# Categorical cross-entropy

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



# Preparing a dataset

```
import pandas as pd
from keras.utils import to_categorical

# Load dataset
df = pd.read_csv('data.csv')
# Turn response variable into labeled codes
df.response = pd.Categorical(df.response)
df.response = df.response.cat.codes
# Turn response variable into one-hot response vector
y = to_categorical(df.response)
```

# One-hot encoding

Label Encoding

Food Name	Categorical #	Calories
Apple	1	95
Chicken	2	231
Broccoli	3	50



One Hot Encoding

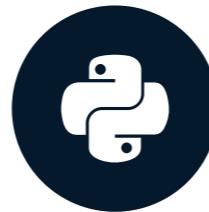
Apple	Chicken	Broccoli	Calories
1	0	0	95
0	1	0	231
0	0	1	50

# **Let's practice!**

**INTRODUCTION TO DEEP LEARNING WITH KERAS**

# Multi-label classification

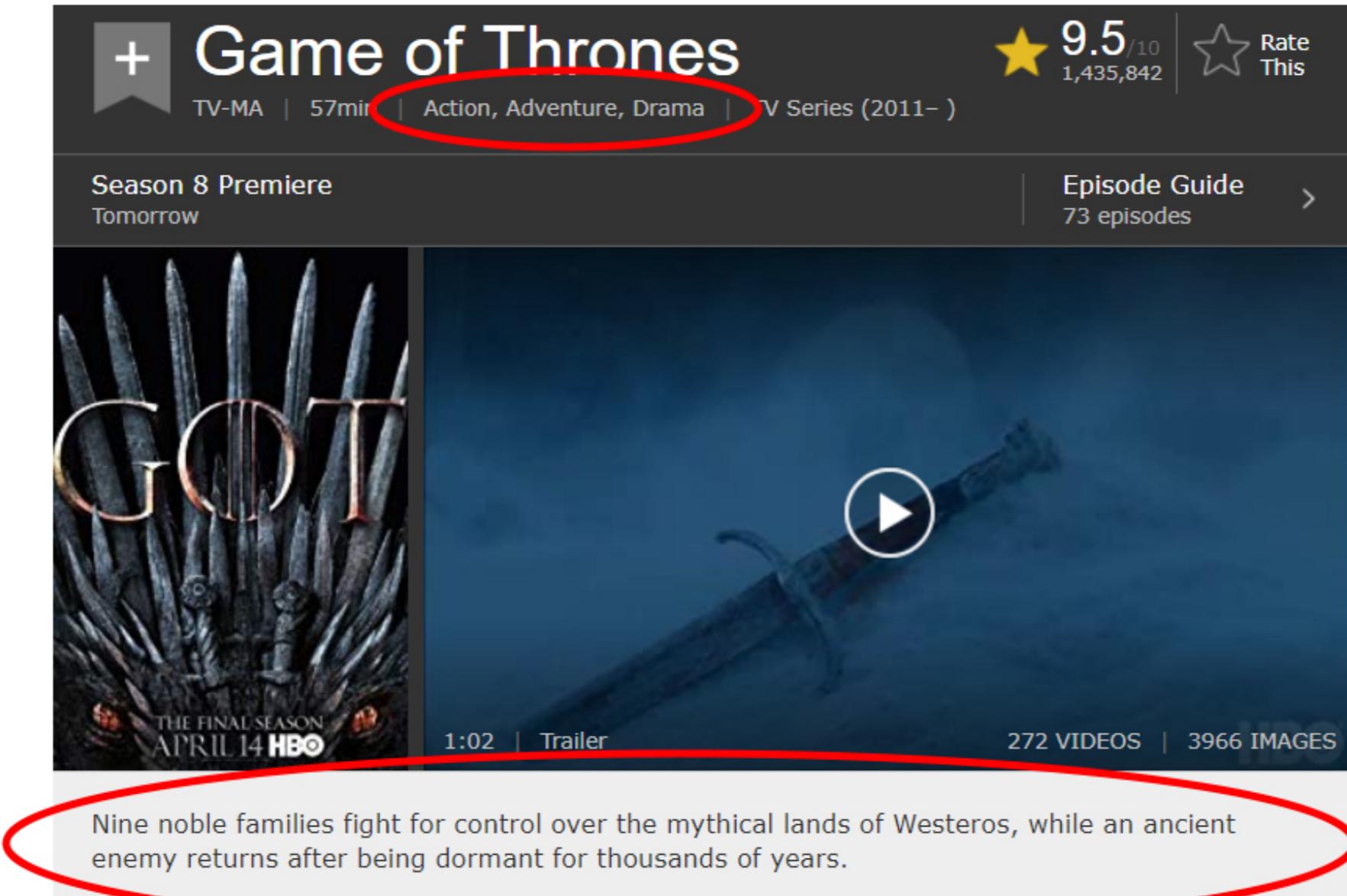
INTRODUCTION TO DEEP LEARNING WITH KERAS



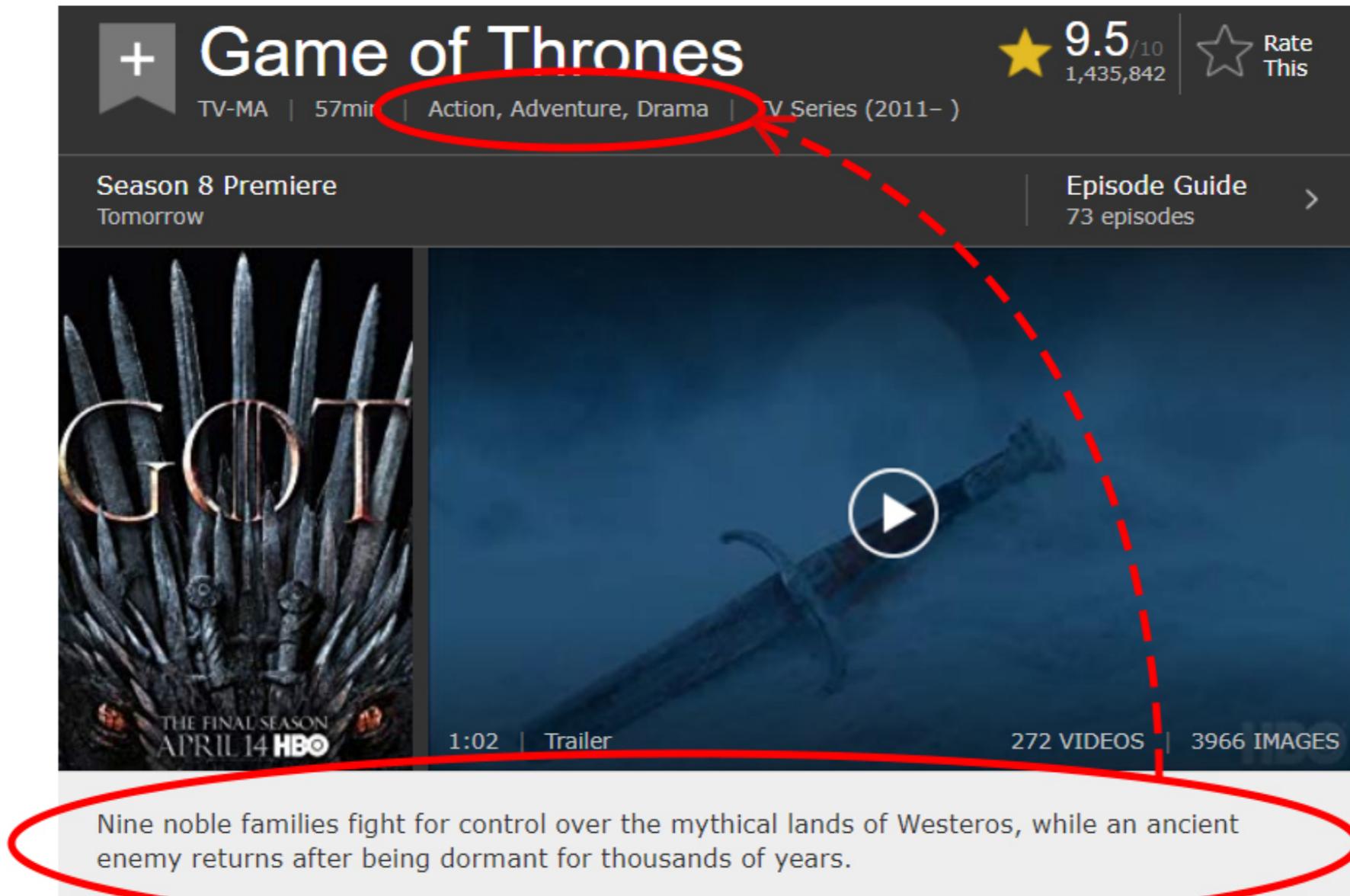
**Miguel Esteban**

Data Scientist & Founder

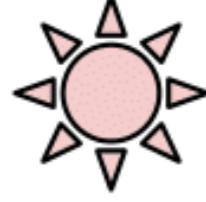
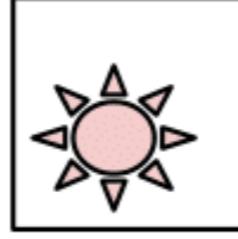
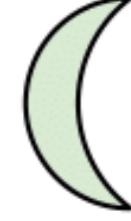
# Real world examples



# Real world examples



## Multi-Class

C = 3	Samples	
		
		
	Labels (t) [0 0 1]   [1 0 0]   [0 1 0]	Labels (t) [1 0 1]   [0 1 0]   [1 1 1]

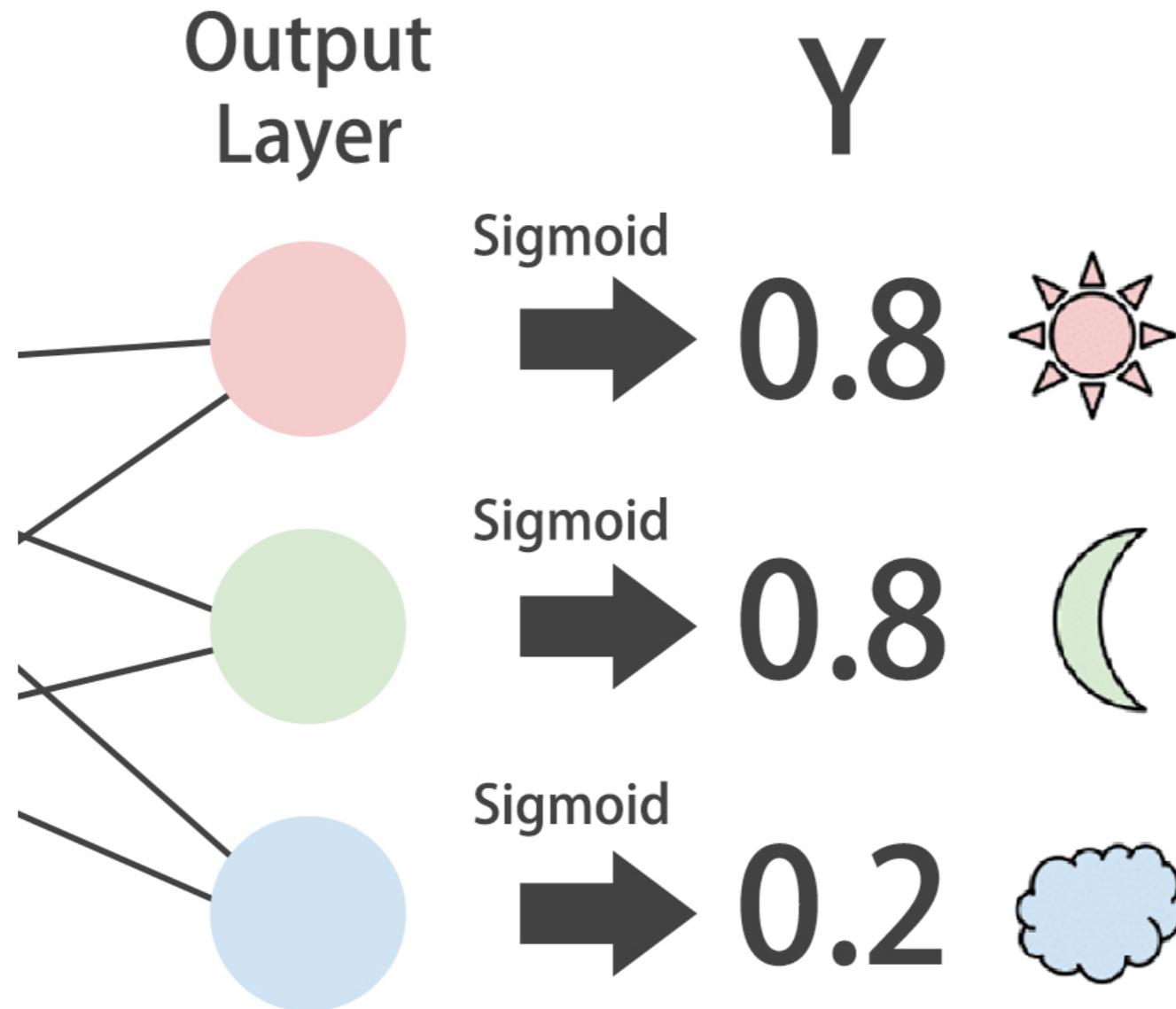
## Multi-Label

<sup>1</sup> [https://gombru.github.io/2018/05/23/cross\\_entropy\\_loss/](https://gombru.github.io/2018/05/23/cross_entropy_loss/)

# The architecture

```
from keras.models import Sequential  
from keras.layers import Dense  
  
# Instantiate model  
model = Sequential()  
# Add input and hidden layers  
model.add(Dense(2, input_shape=(1,)))  
# Add an output layer for the 3 classes and sigmoid activation  
model.add(Dense(3, activation='sigmoid'))
```

# Sigmoid outputs

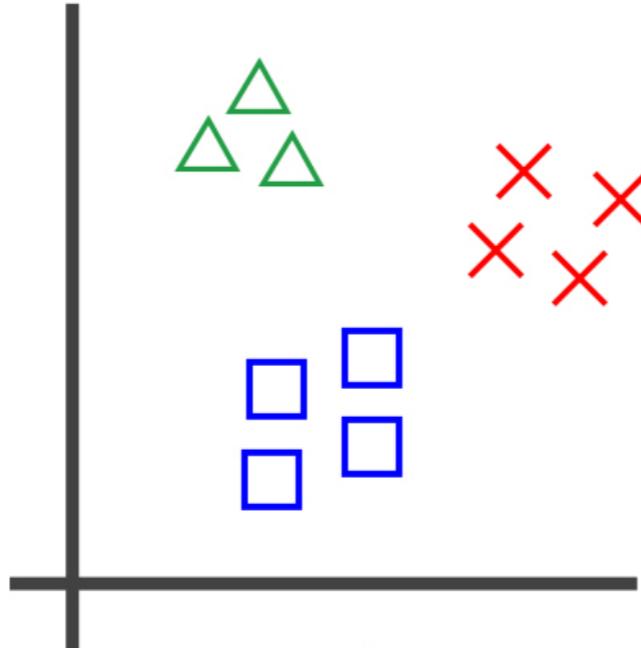


```
# Compile the model with binary crossentropy  
model.compile(optimizer='adam', loss='binary_crossentropy')
```

```
# Train your model, recall validation_split  
model.fit(X_train, y_train,  
          epochs=100,  
          validation_split=0.2)
```

```
Train on 1260 samples, validate on 280 samples  
Epoch 1/100  
1260/1260 [=====] - 0s 285us/step  
- loss: 0.7035 - acc: 0.6690 - val_loss: 0.5178 - val_acc: 0.7714  
...
```

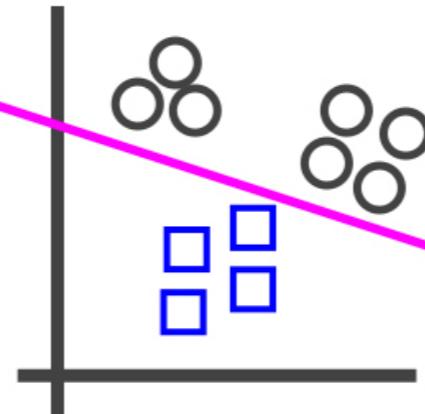
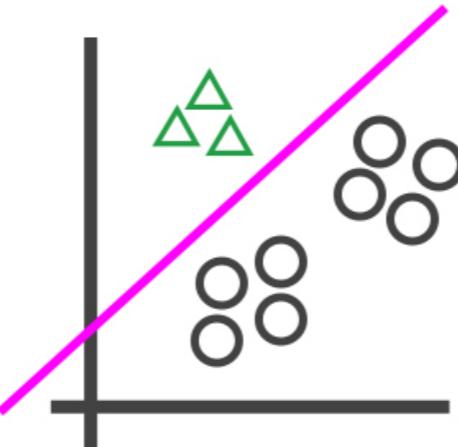
## One-vs-rest



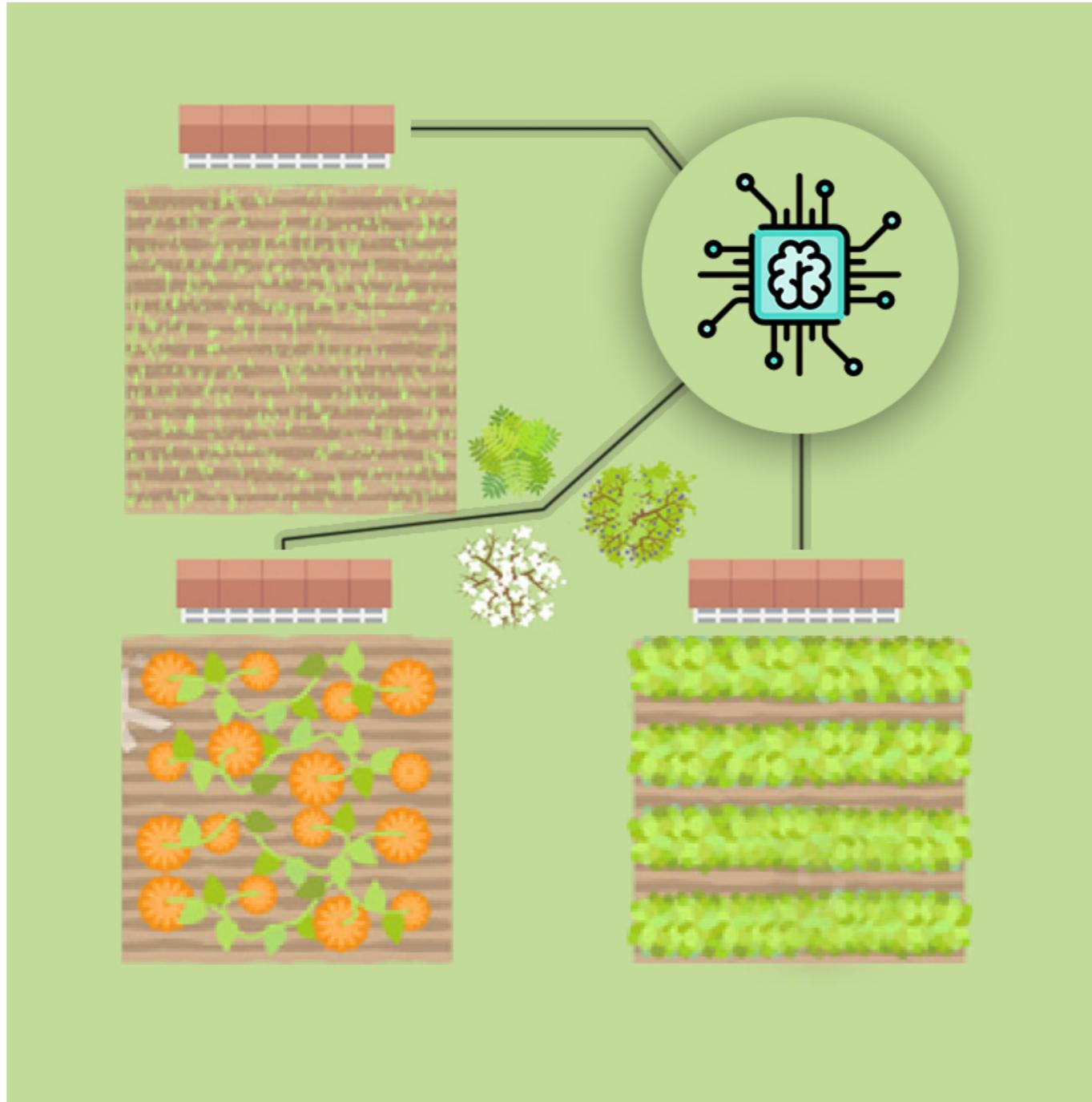
class 1: ▲

class 2: □

class 3: ✕



# An irrigation machine



# An irrigation machine

## Sensor measurements

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	4.0	3.0	4.0	4.0	1.0	1.0	4.0	0.0	3.0	3.0	2.0	2.0	3.0	1.0	4.0	2.0	2.0	2.0	1.0	4.0
1	1.0	1.0	6.0	3.0	2.0	3.0	4.0	5.0	1.0	3.0	3.0	2.0	3.0	2.0	2.0	4.0	0.0	1.0	2.0	4.0
2	1.0	5.0	7.0	6.0	4.0	0.0	0.0	6.0	0.0	1.0	1.0	3.0	4.0	2.0	1.0	0.0	4.0	1.0	1.0	3.0
3	0.0	1.0	3.0	3.0	7.0	1.0	2.0	2.0	0.0	4.0	3.0	2.0	4.0	2.0	0.0	2.0	3.0	4.0	1.0	2.0
4	1.0	5.0	2.0	2.0	1.0	0.0	3.0	3.0	1.0	1.0	5.0	1.0	1.0	2.0	2.0	4.0	3.0	3.0	0.0	5.0

## Parcels to water

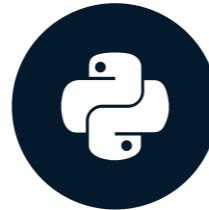
	0	1	2
0	0	0	0
1	1	1	1
2	1	1	0
3	1	1	1
4	0	0	0

# **Let's practice!**

**INTRODUCTION TO DEEP LEARNING WITH KERAS**

# Keras callbacks

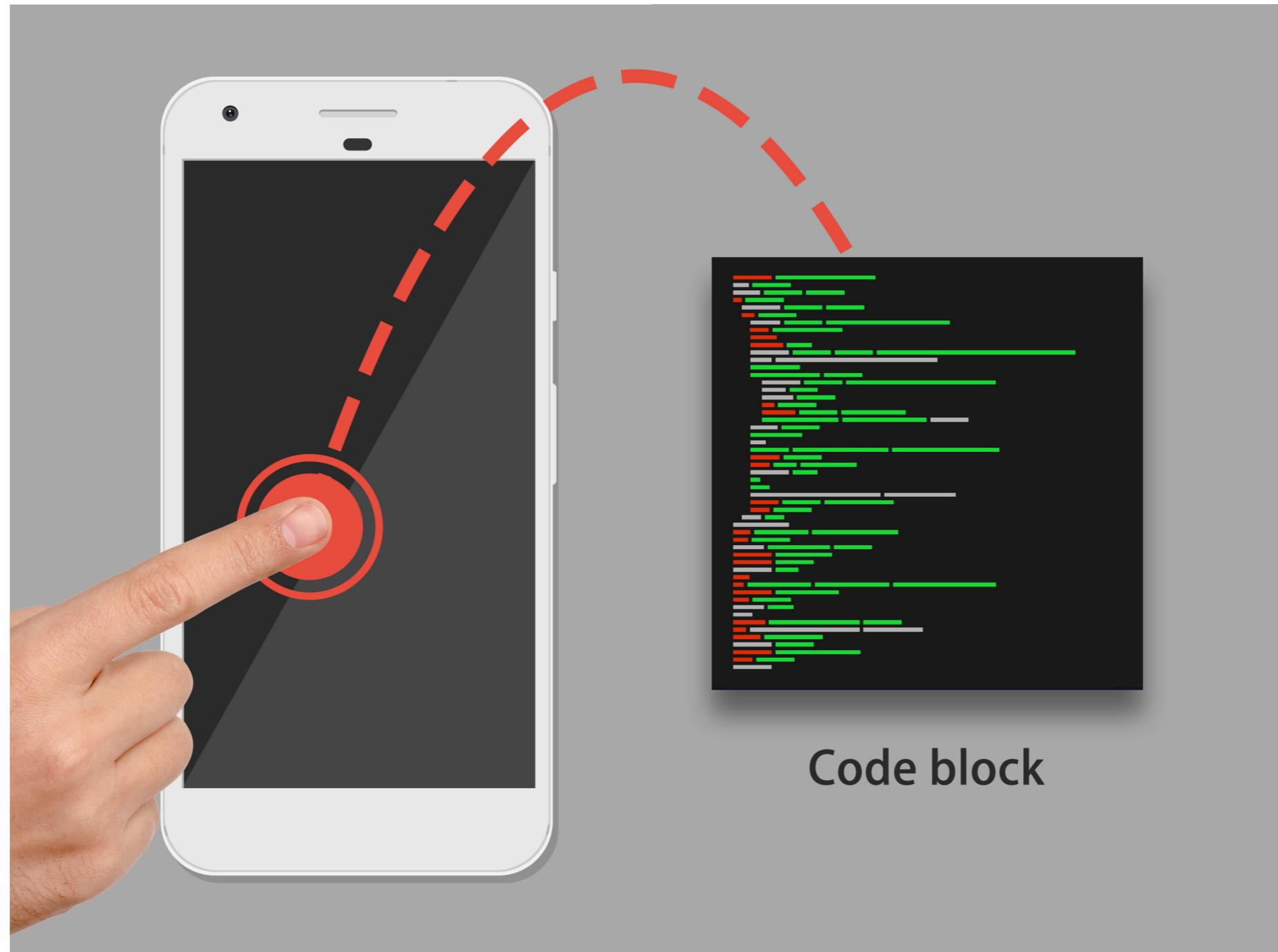
INTRODUCTION TO DEEP LEARNING WITH KERAS



**Miguel Esteban**

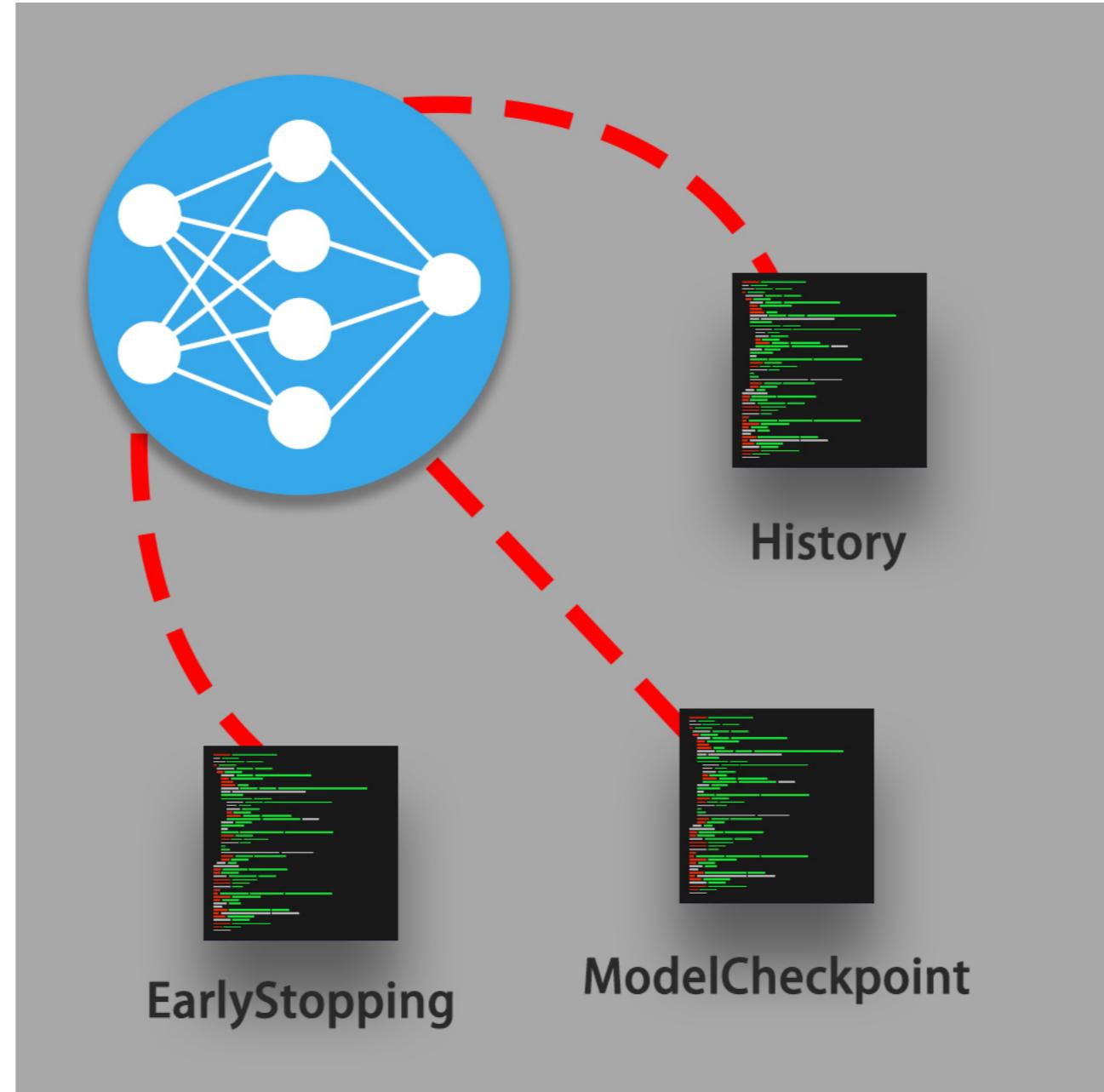
Data Scientist & Founder

# What is a callback?



Code block

# Callbacks in Keras



# A callback you've been missing

```
# Training a model and saving its history  
history = model.fit(X_train, y_train,  
                      epochs=100,  
                      metrics=['accuracy'])  
print(history.history['loss'])
```

```
[0.6753975939750672, ..., 0.3155936544282096]
```

```
print(history.history['acc'])
```

```
[0.7030952412741525, ..., 0.8604761900220599]
```

# A callback you've been missing

```
# Training a model and saving its history
history = model.fit(X_train, y_train,
                     epochs=100,
                     validation_data=(X_test, y_test),
                     metrics=['accuracy'])
print(history.history['val_loss'])
```

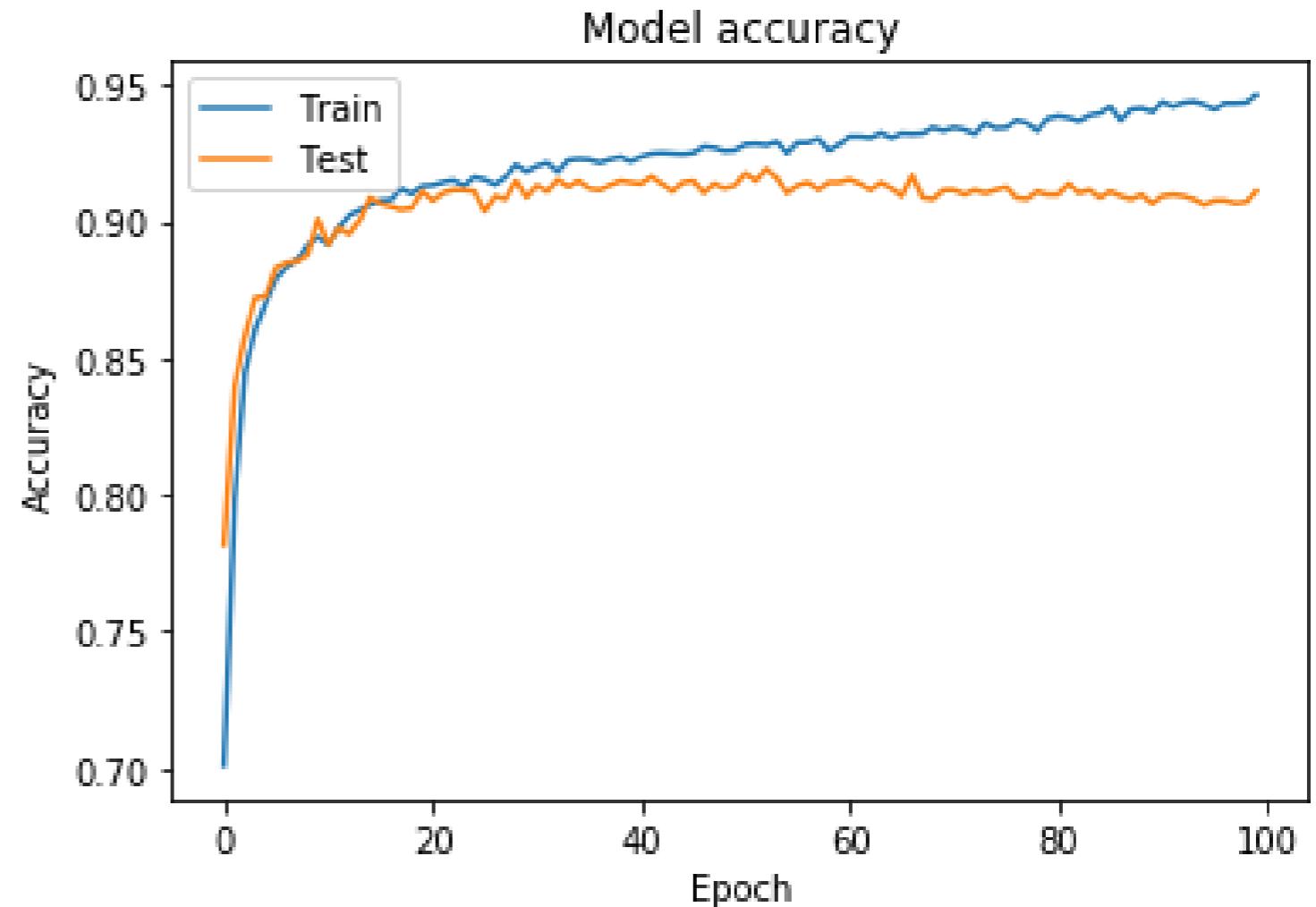
```
[0.7753975939750672, ..., 0.4155936544282096]
```

```
print(history.history['val_acc'])
```

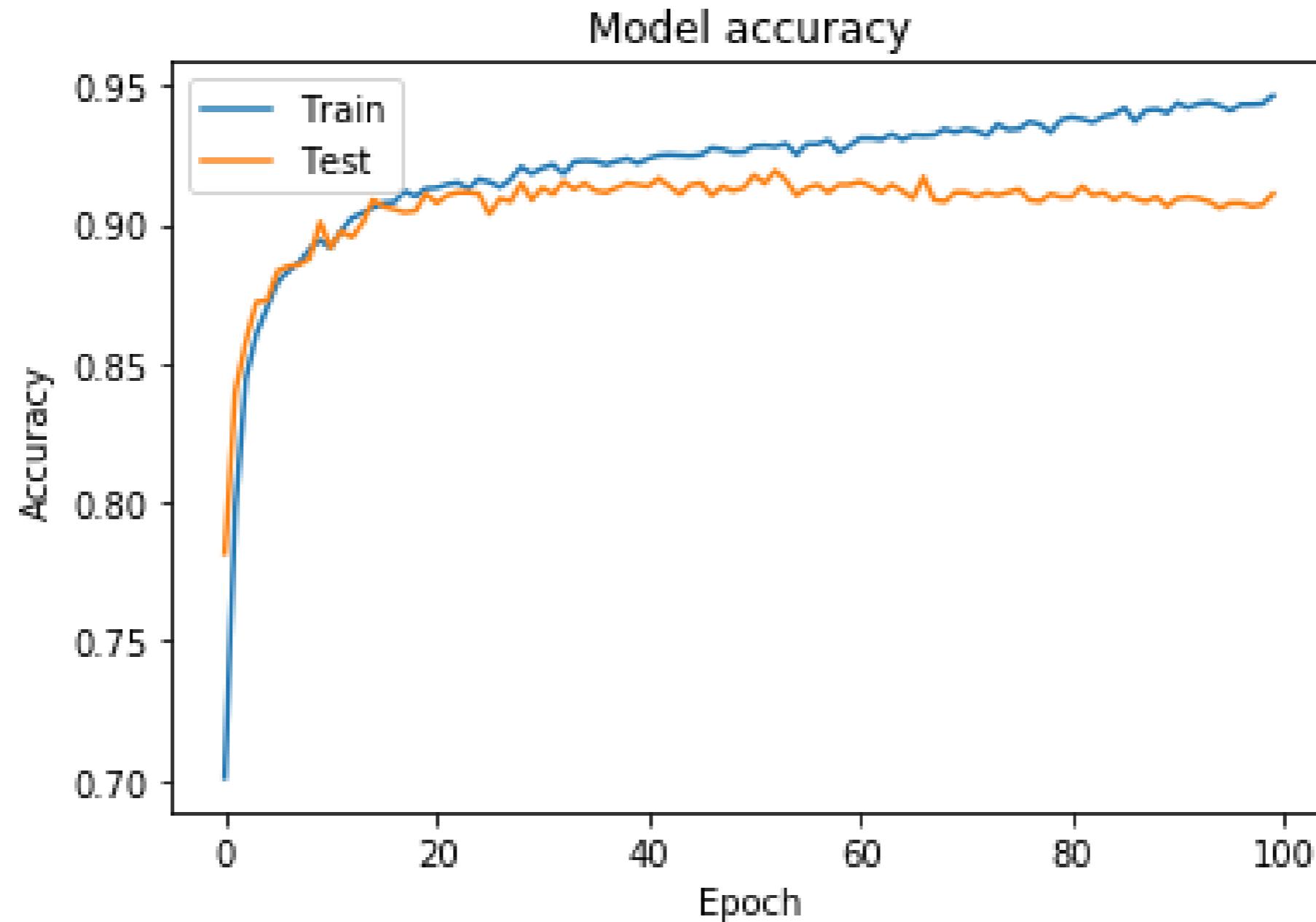
```
[0.6030952412741525, ..., 0.7604761900220599]
```

# History plots

```
# Plot train vs test accuracy per epoch  
plt.figure()  
  
# Use the history metrics  
plt.plot(history.history['acc'])  
plt.plot(history.history['val_acc'])  
  
# Make it pretty  
plt.title('Model accuracy')  
plt.ylabel('Accuracy')  
plt.xlabel('Epoch')  
plt.legend(['Train', 'Test'])  
plt.show()
```



# History plots



# Early stopping

```
# Import early stopping from keras callbacks
from keras.callbacks import EarlyStopping

# Instantiate an early stopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=5)

# Train your model with the callback
model.fit(X_train, y_train, epochs=100,
           validation_data=(X_test, y_test),
           callbacks = [early_stopping])
```

# Model checkpoint

```
# Import model checkpoint from keras callbacks
from keras.callbacks import ModelCheckpoint

# Instantiate a model checkpoint callback
model_save = ModelCheckpoint('best_model.hdf5',
                             save_best_only=True)

# Train your model with the callback
model.fit(X_train, y_train, epochs=100,
           validation_data=(X_test, y_test),
           callbacks = [model_save])
```

# **Let's practice!**

**INTRODUCTION TO DEEP LEARNING WITH KERAS**