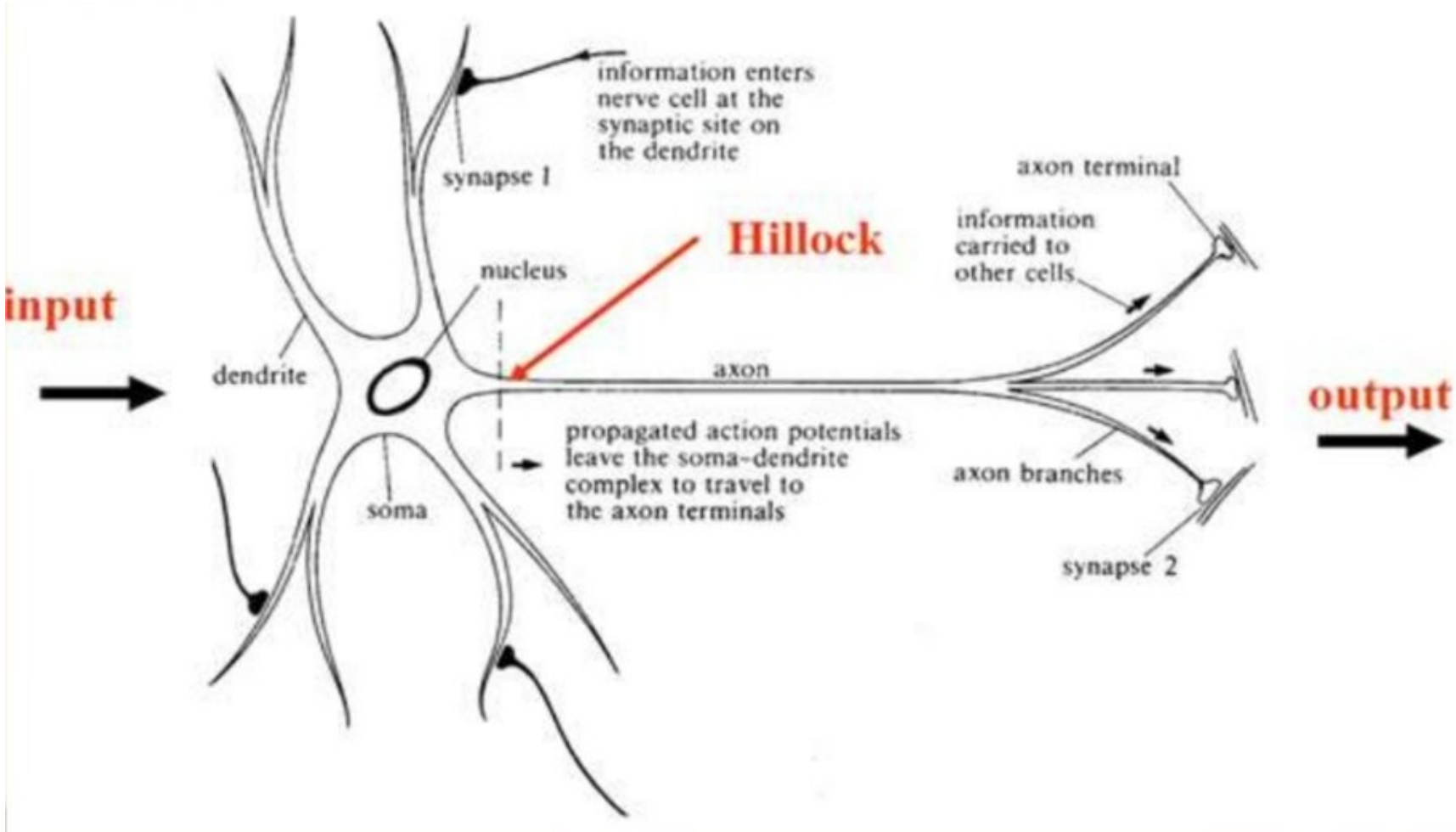
ARTIFICIAL NEURAL NETWORKS

INTRODUCTION

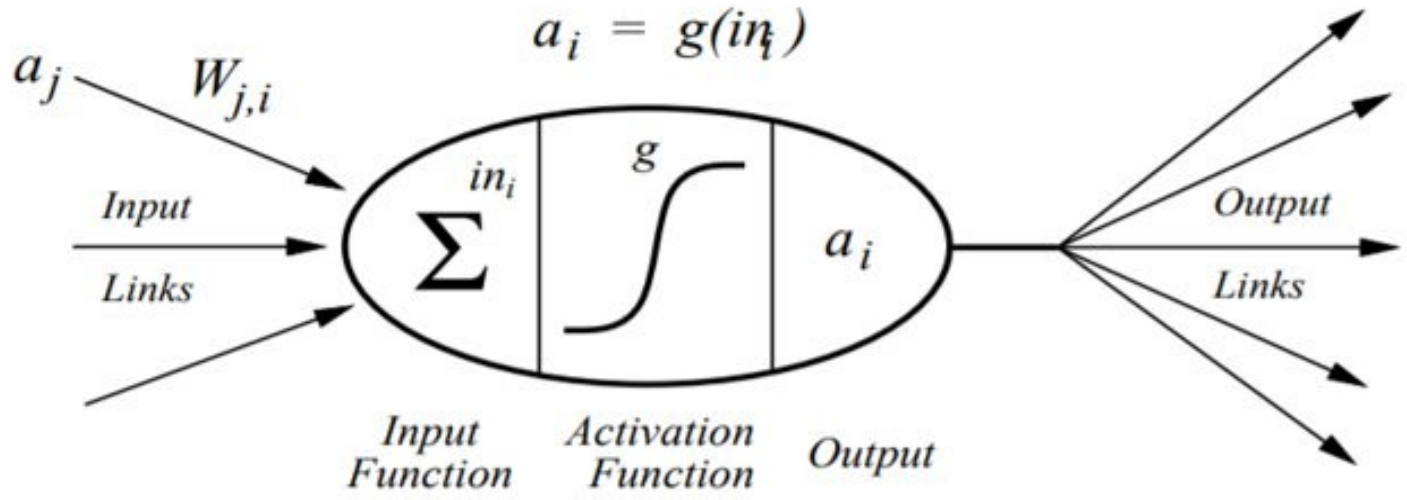
- WHAT IS PYTHON?
- WHAT CAN YOU DO WITH IT?
- WHY IS IT SO POPULAR?

PYTHON – THE TOP 3 QUESTIONS

BIOLOGICAL INSPIRATION



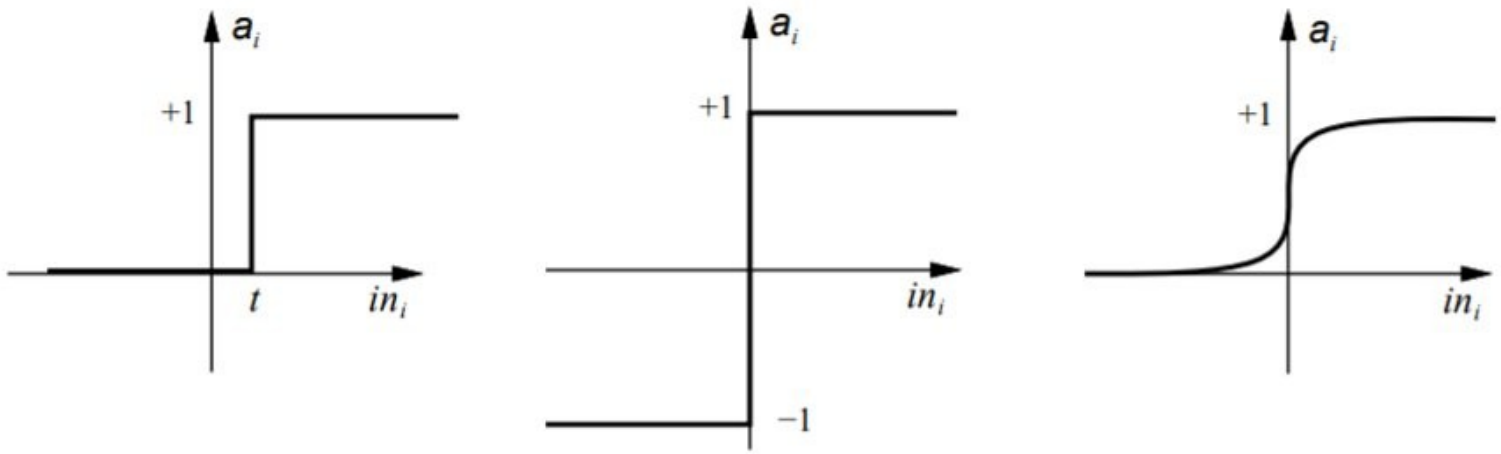
THE PERCEPTRON



$$a_i = g(\sum_j W_{j,i} a_j)$$

WHAT IS THE ACTIVATION FUNCTION?

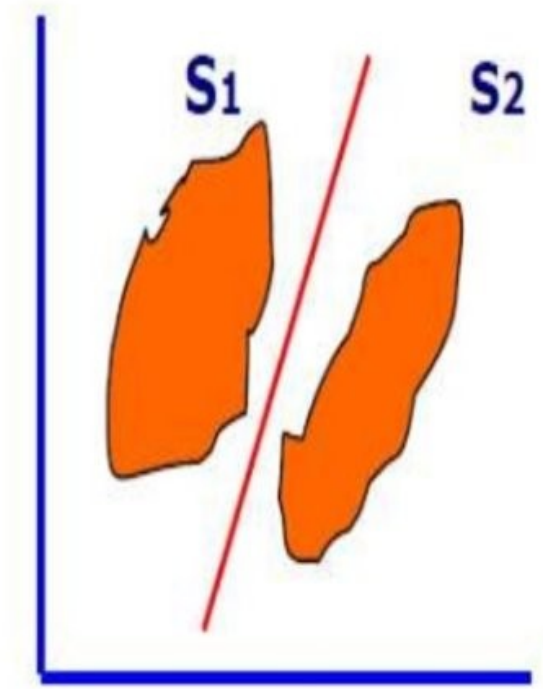
- A simple function that determines the output of this perceptron i.e. when will this perceptron will "fire"

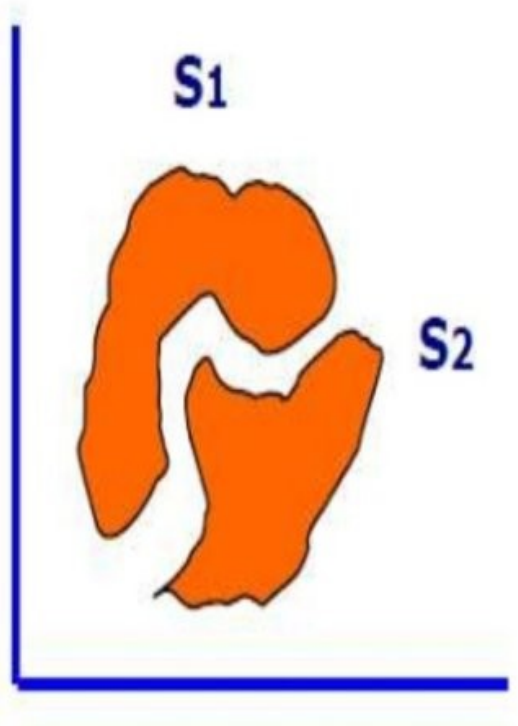


(a) Step function (b) Sign function (c) Sigmoid function

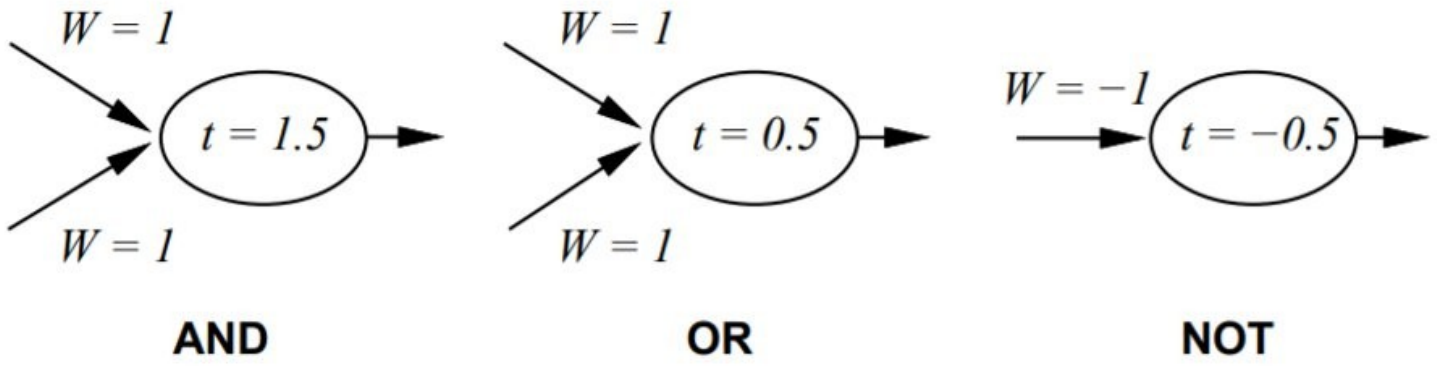
THE PERCEPTRON

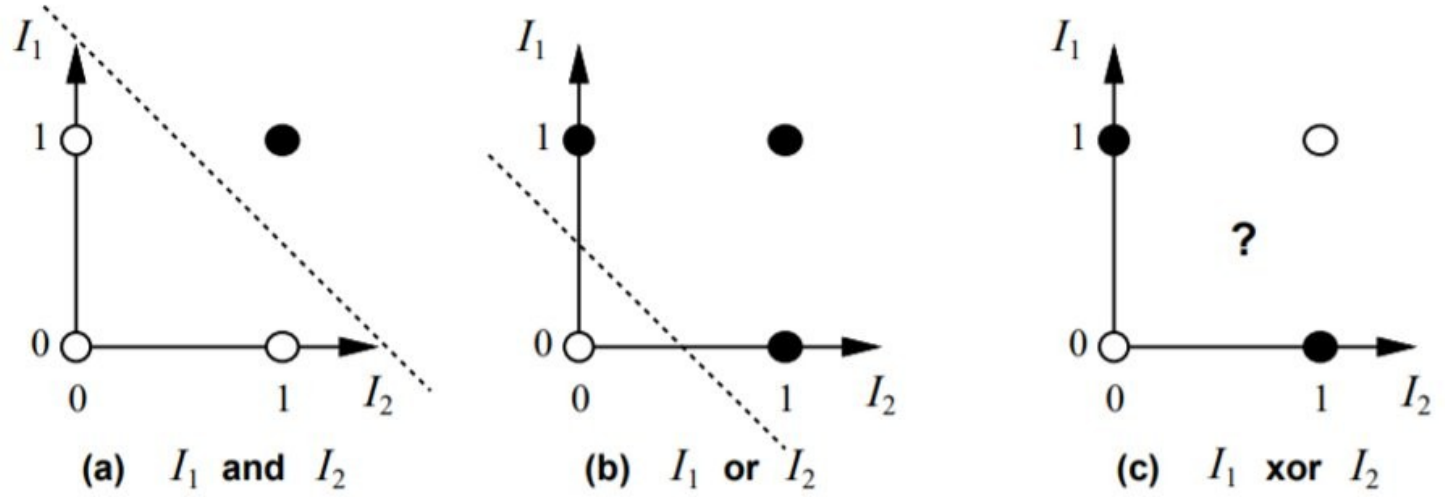
- A single perceptron can be tuned to classify any linearly separable set of inputs
- https://playground.tensorflow.org/





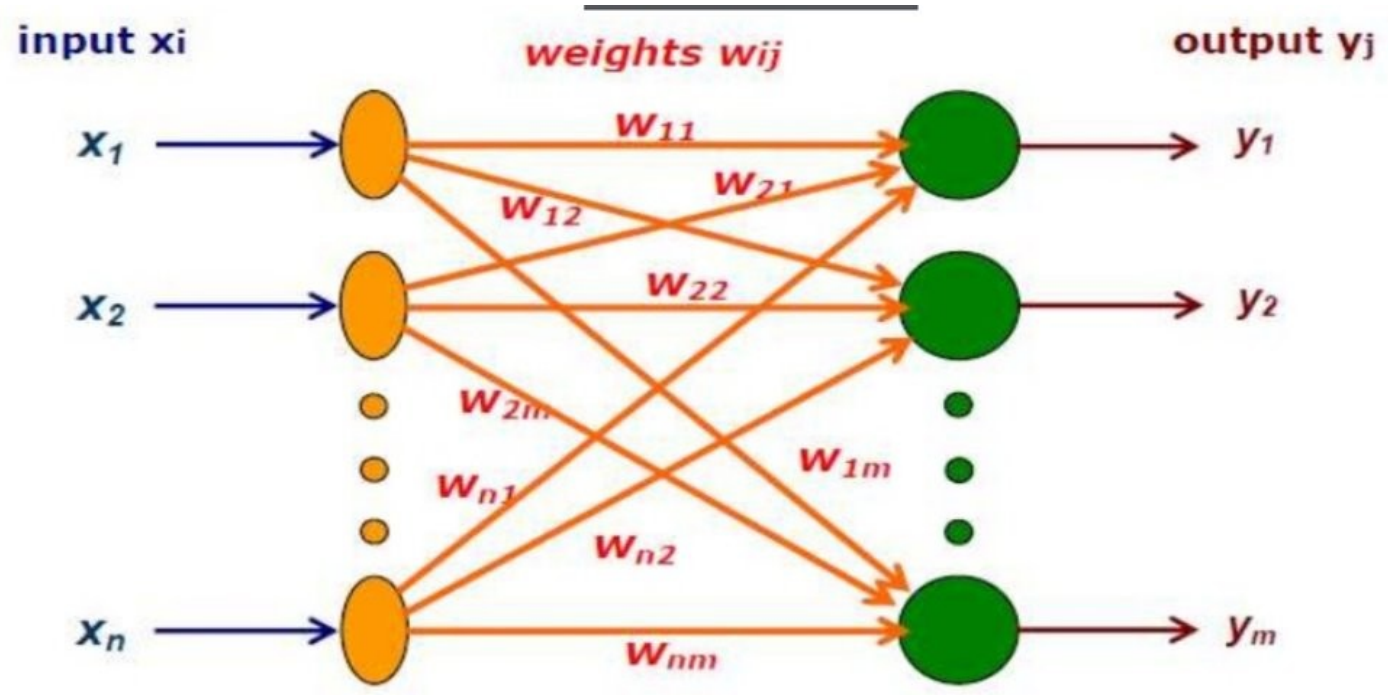
A SINGLE PERCEPTRON





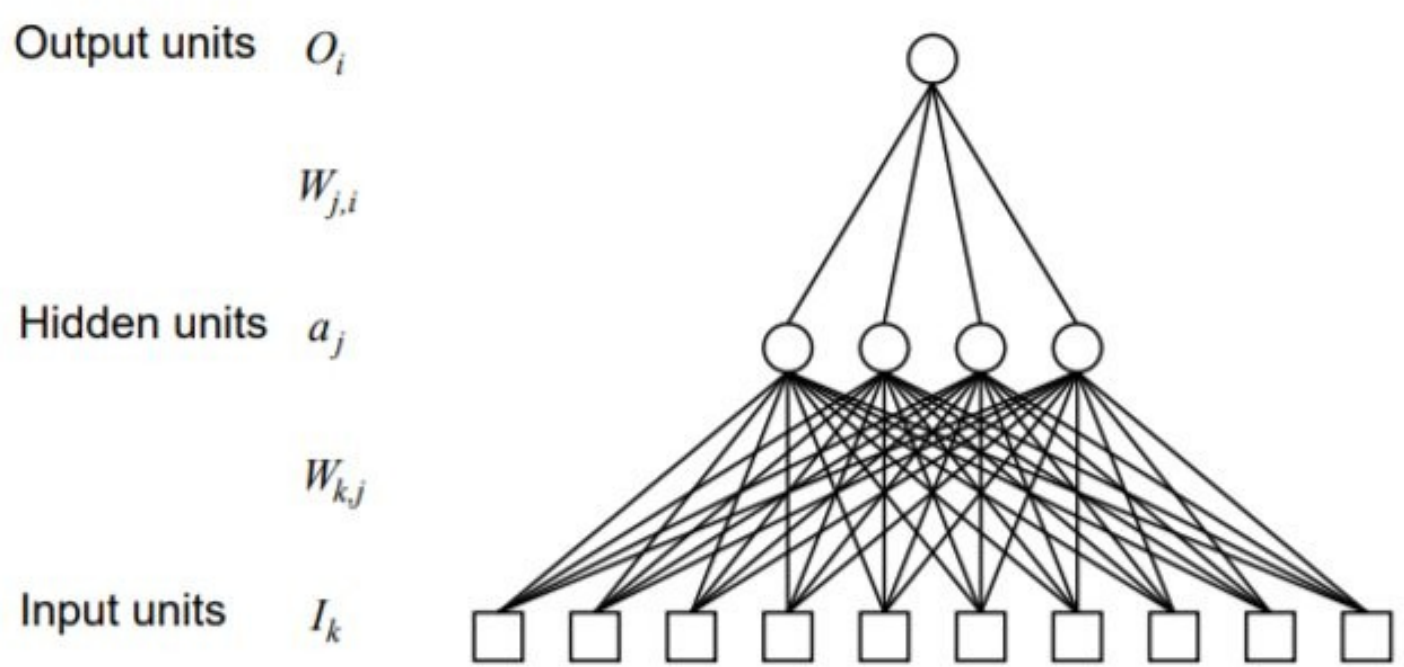
A SINGLE LAYER PERCEPTRON

- By combining many perceptrons, the overall output can match almost any complex pattern
- A Single Layer Perceptron has a single input layer and a single output layer



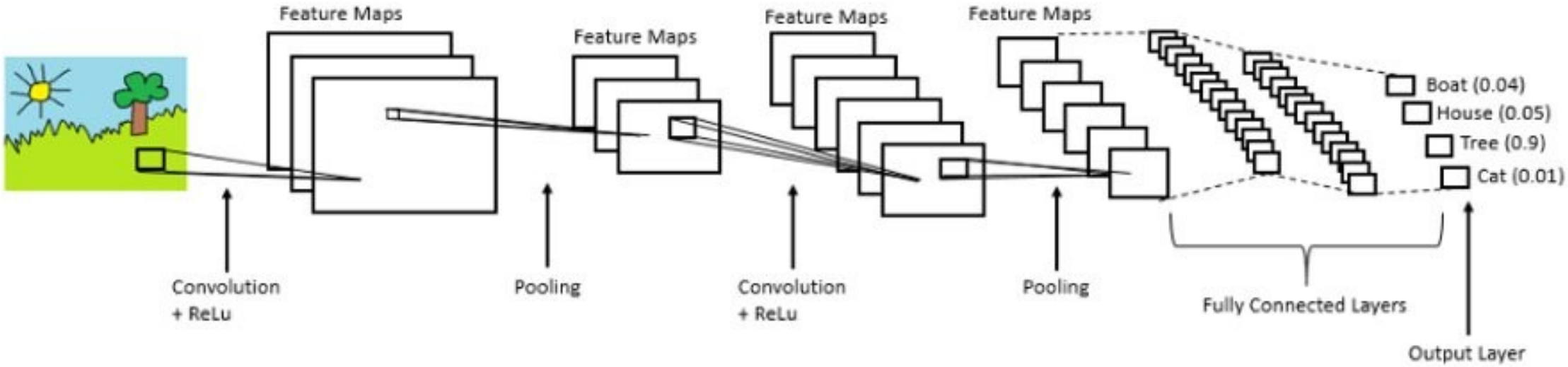
MULTI-LAYER PERCEPTRON

- The classic "Artificial Neural Network"
- Here we have more layers called "hidden layers" between the input and output layers



CONVOLUTIONAL NEURAL NETWORK

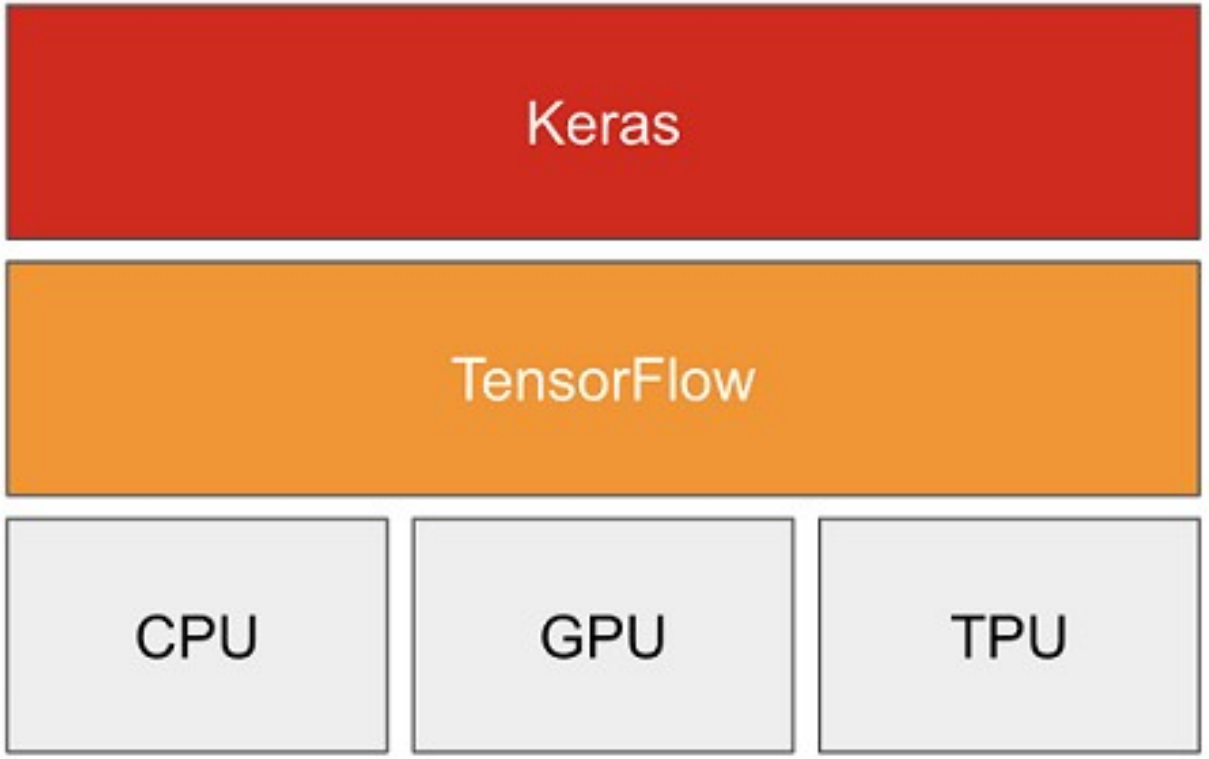
- A more complex architecture often used in image processing
- This pre-processes the input images with filtering layers so that the input to an MLP is not just "pixel values" from an image



PYTHON AND NEURAL NETWORKS

- To create these Neural Network structures in code we need efficient libraries that give us
  - Flexibility to create whatever network architecture we need
  - Ease of use to abstract us from low-level coding details
- Python is well suited to this, and so a number of libraries have become very popular

KERAS & TENSORFLOW



Deep learning development: layers, models, optimizers, losses, metrics...

Tensor manipulation infrastructure: tensors, variables, automatic differentiation, distribution...

Hardware: execution

KERAS FUNCTIONAL API

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

PYTORCH

- Commonly uses a more object-oriented coding style than Keras

```python
import torch
import torch.nn as nn

class TwoLayerNet(nn.Module):
    """
    In the constructor we instantiate two nn.Linear modules and assign them as
    member variables.
    """
    def __init__(self, input_size, hidden_layers, output_size):
        super(TwoLayerNet, self).__init__()
        self.l1 = nn.Linear(input_size, hidden_layers)
        self.relu = nn.ReLU()
        self.l2 = nn.Linear(hidden_layers, output_size)

    def forward(self, x):
        """
        In the forward function we accept a Tensor of input data and we must return
        a Tensor of output data. We can use Modules defined in the constructor as
        well as arbitrary (differentiable) operations on Tensors.
        """
        y_pred = self.l1(x)
        y_pred = self.relu(y_pred)
        y_pred = self.l2(y_pred)

        return y_pred
```

QUESTIONS