

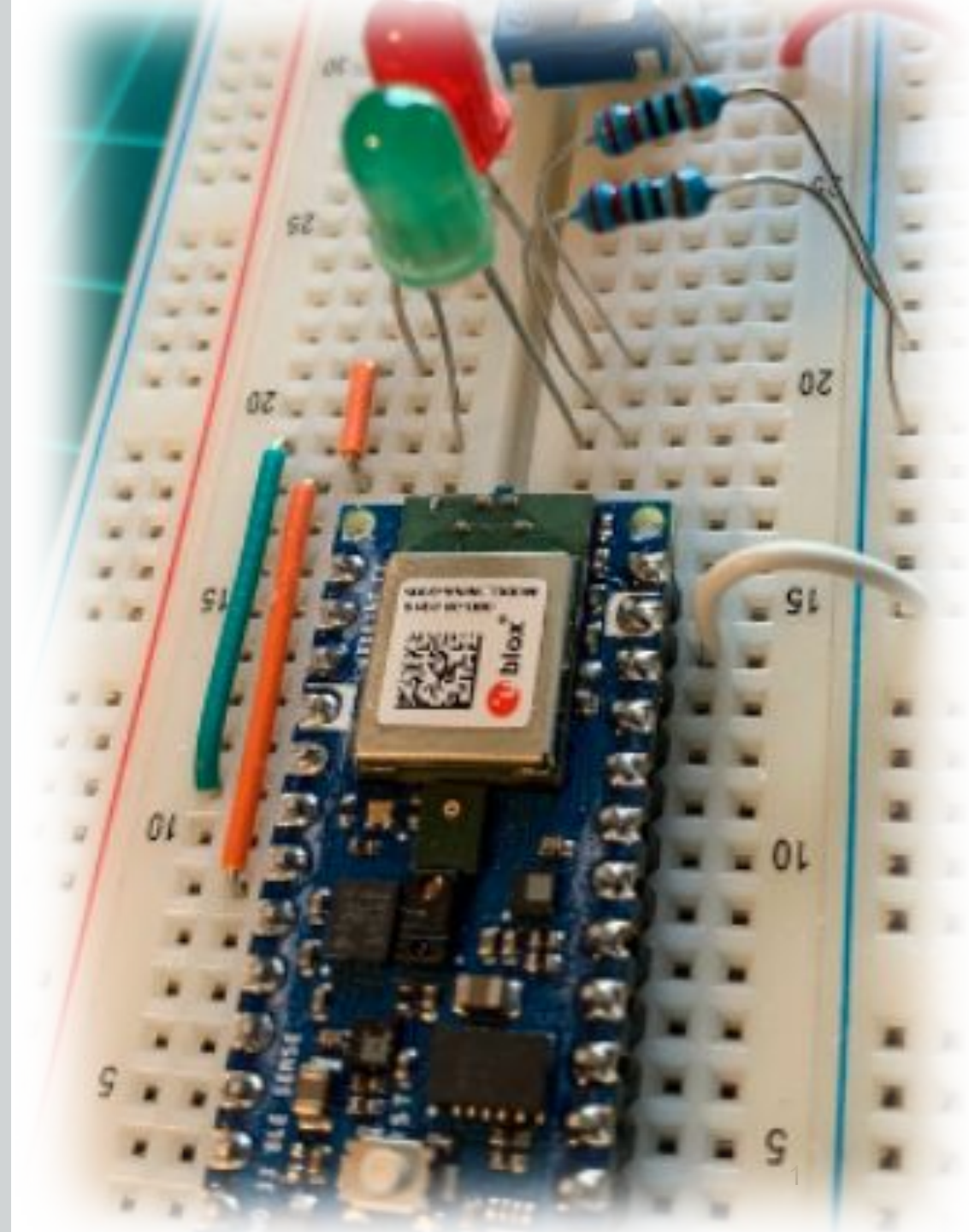
# IESTI01 – TinyML

## Embedded Machine Learning

6. The Building Blocks of  
Deep Learning – Part B
- Classification

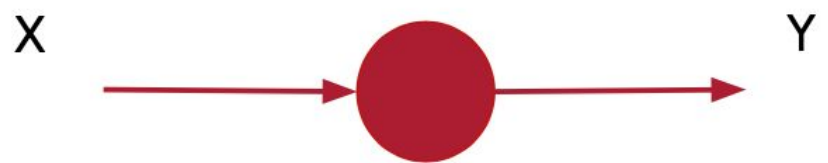


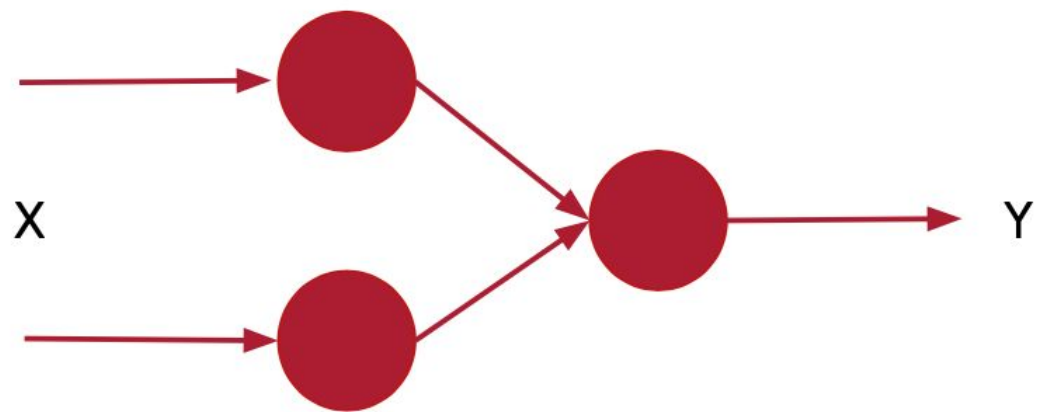
Prof. Marcelo Rovai  
UNIFEI

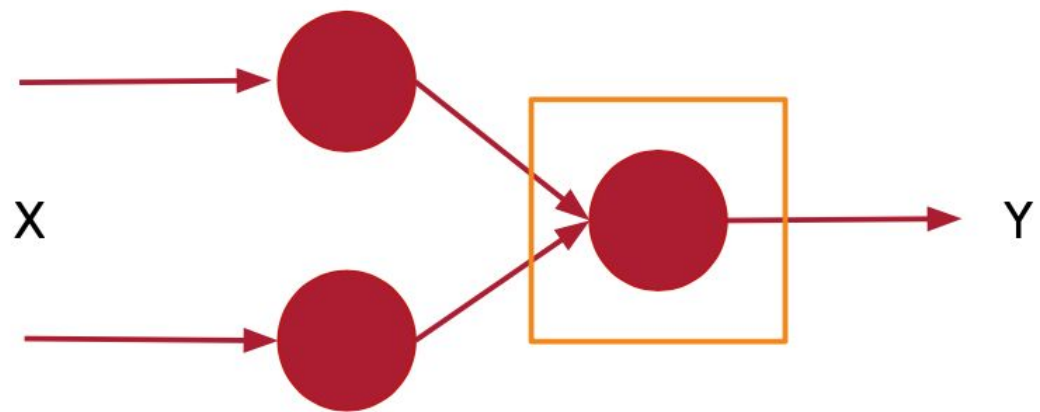


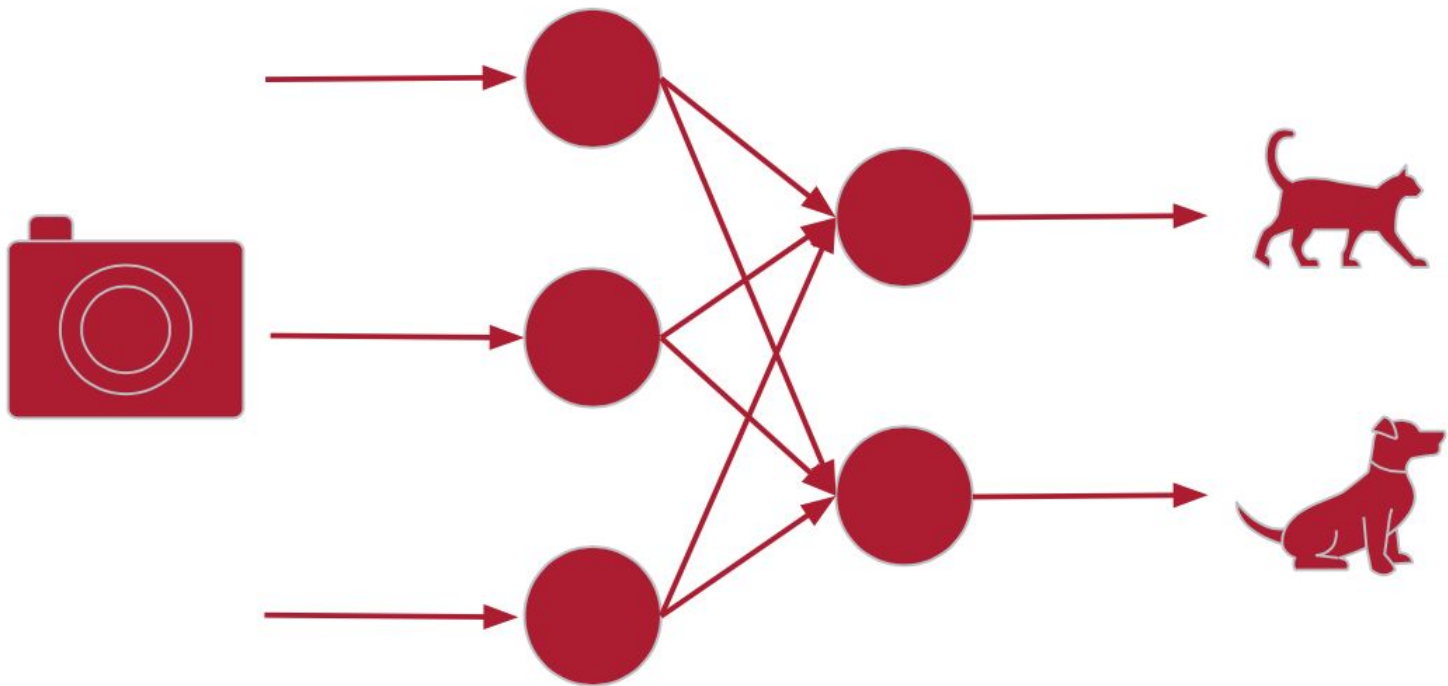
# Going Further

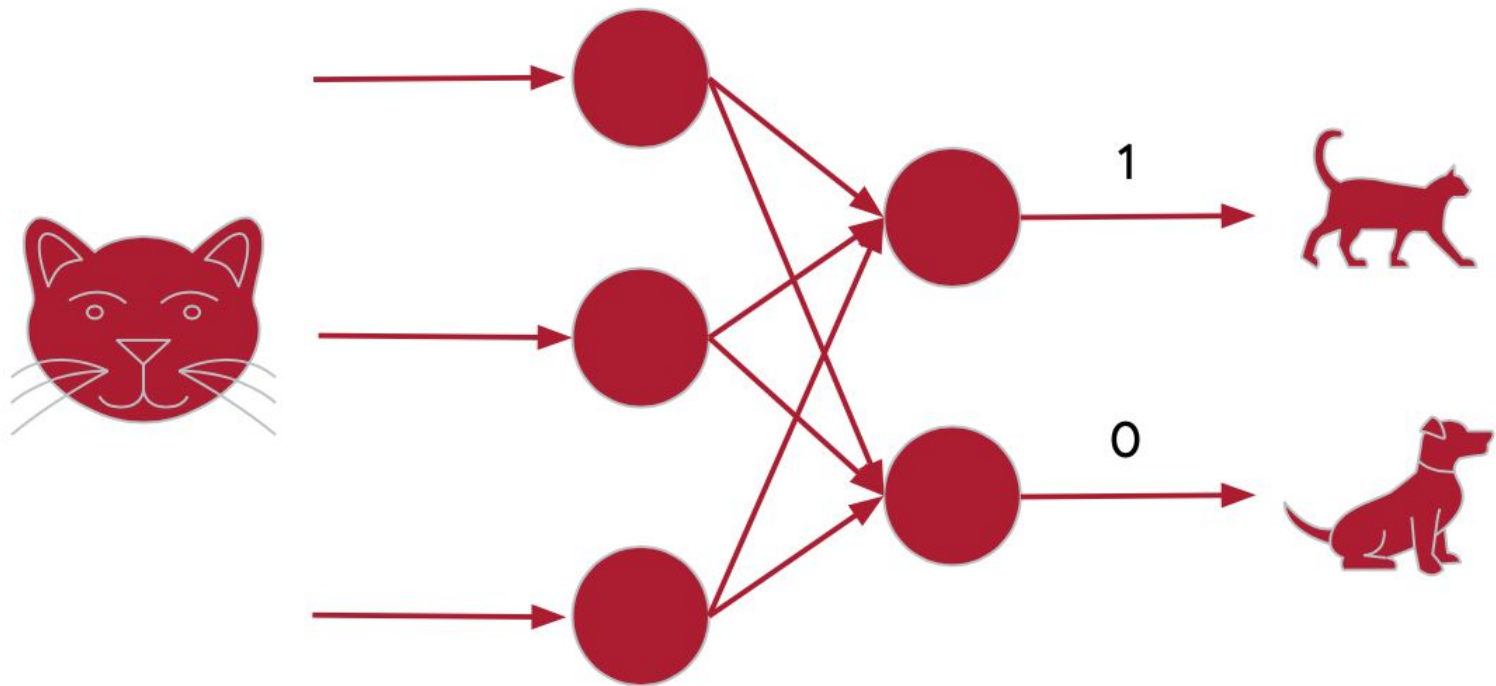
From regression to classification

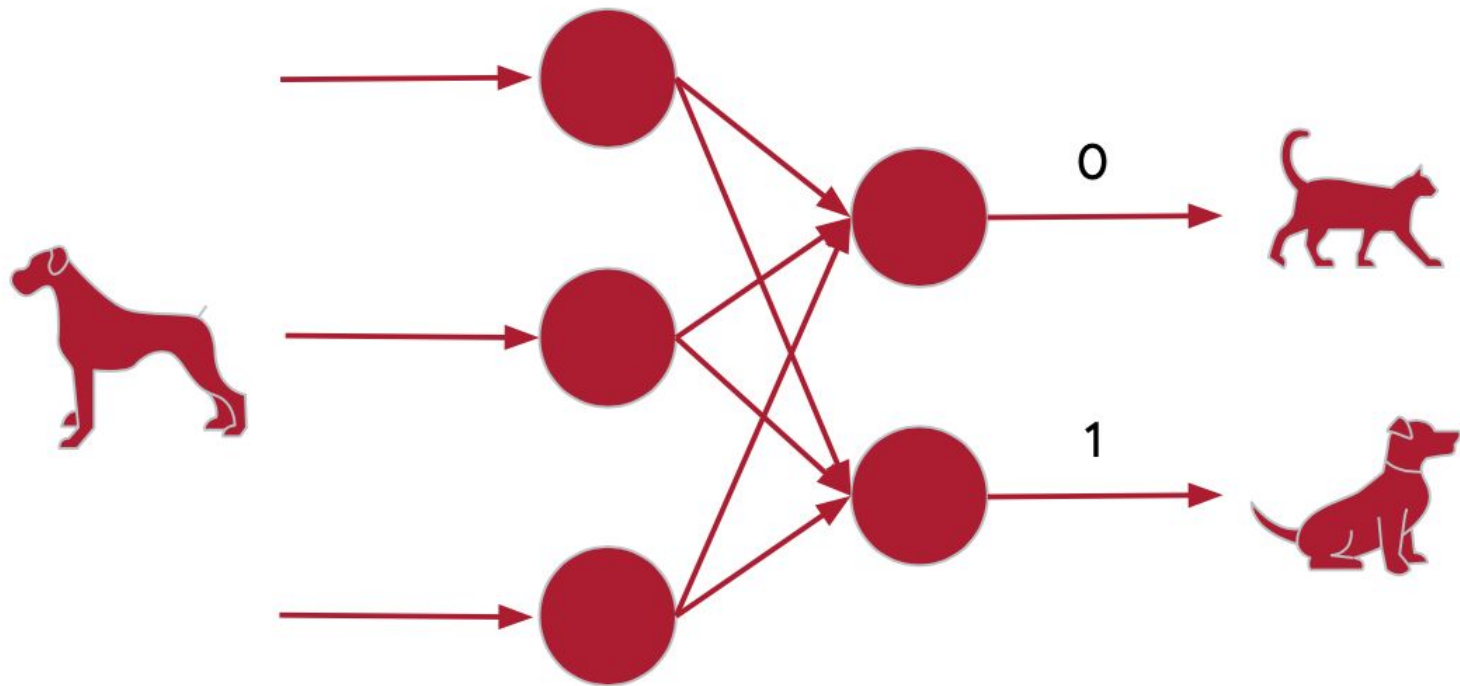












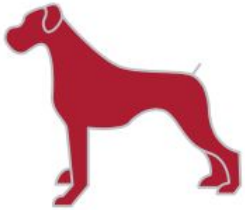


Data

Label



[ 1, 0 ]



[ 0, 1 ]

0 [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

1 [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]

2 [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]

3 [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]

4 [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]

5 [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]

6 [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]

7 [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]

8 [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]

9 [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]

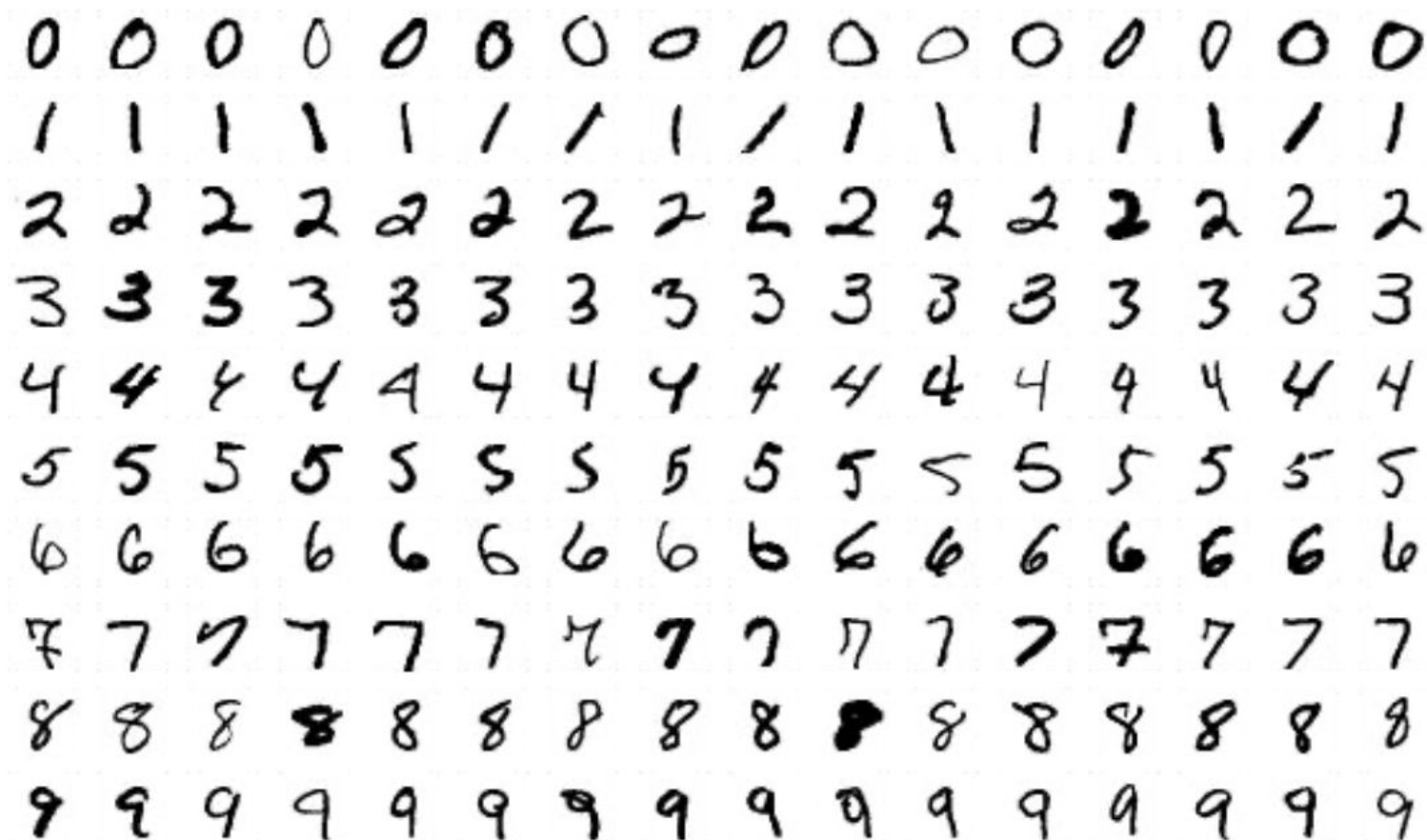
```
import tensorflow as tf
```

```
data = tf.keras.datasets.mnist  
(training_images, training_labels), (val_images, val_labels) = data.load_data()
```

```
training_images = training_images / 255.0
```

```
val_images = val_images / 255.0
```

```
model = tf.keras.models.Sequential(  
    [tf.keras.layers.Flatten(input_shape=(28,28)),  
     tf.keras.layers.Dense(20, activation=tf.nn.relu),  
     tf.keras.layers.Dense(10, activation=tf.nn.softmax)])
```



60,000 Labelled Training Examples  
10,000 Labelled Validation Examples

```
import tensorflow as tf

data = tf.keras.datasets.mnist
(training_images, training_labels), (val_images, val_labels) = data.load_data()

training_images = training_images / 255.0
val_images = val_images / 255.0

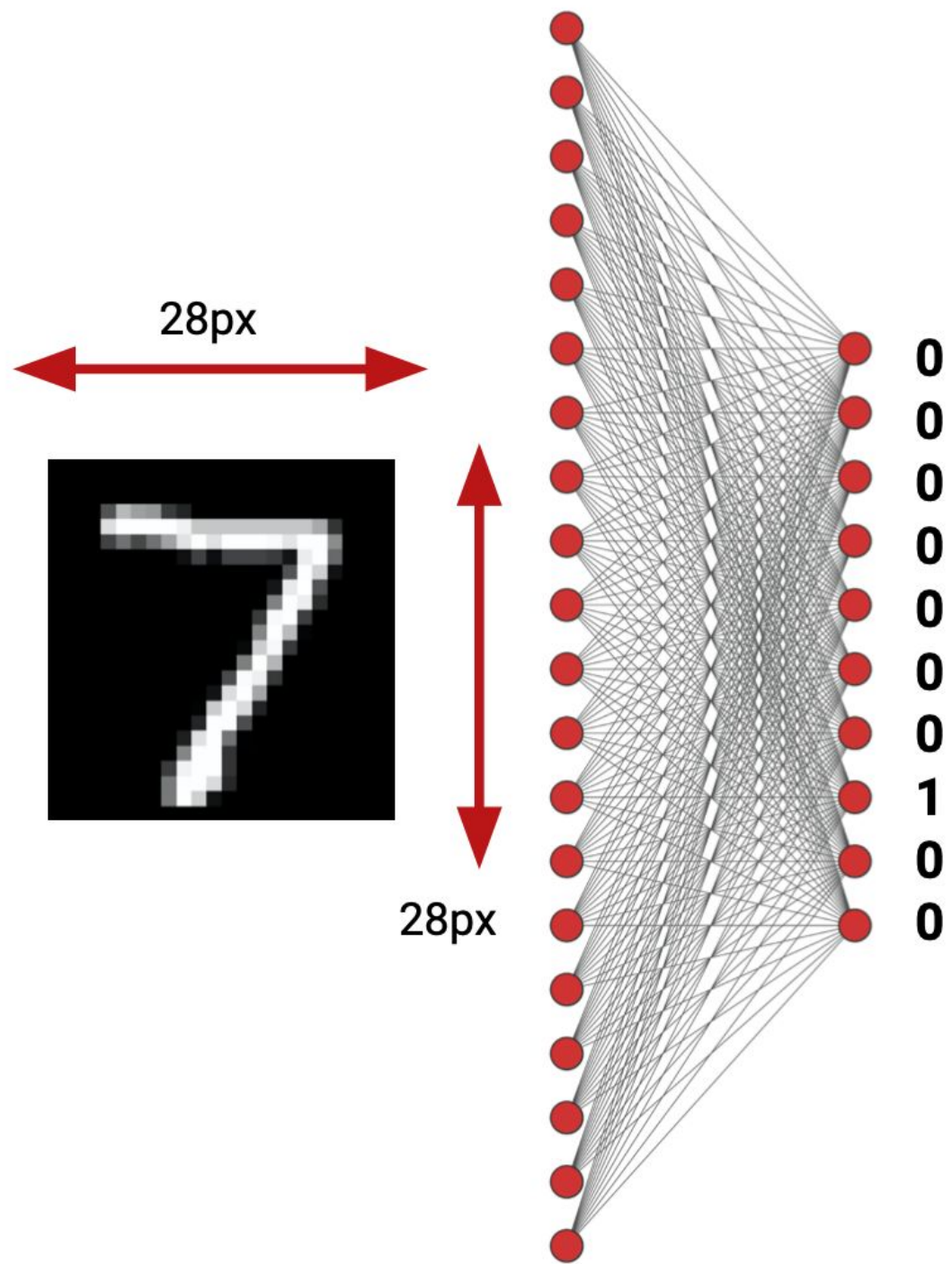
model = tf.keras.models.Sequential(
    [tf.keras.layers.Flatten(input_shape=(28,28)),
     tf.keras.layers.Dense(20, activation=tf.nn.relu),
     tf.keras.layers.Dense(10, activation=tf.nn.softmax)])
```

```
import tensorflow as tf

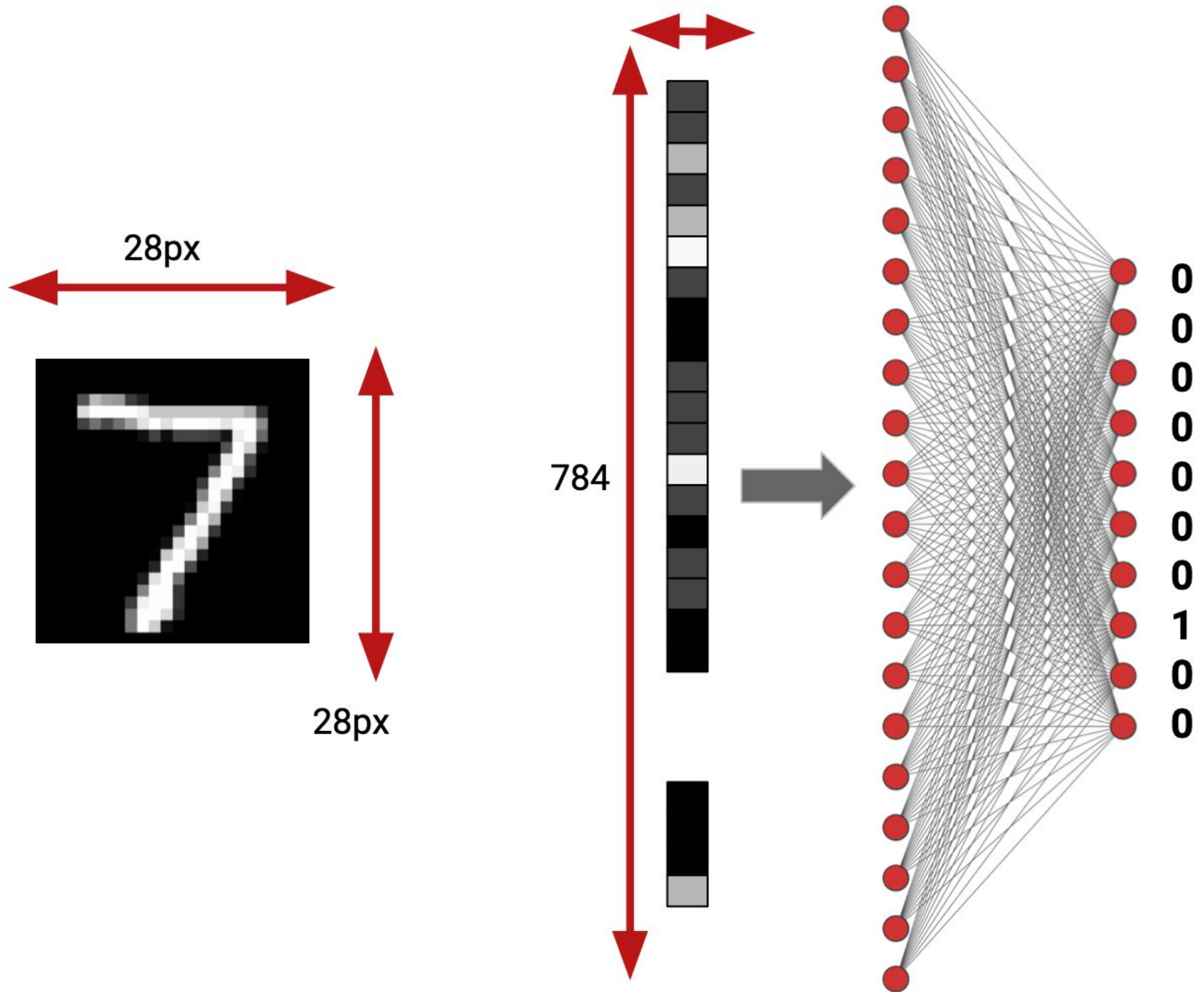
data = tf.keras.datasets.mnist
(training_images, training_labels), (val_images, val_labels) = data.load_data()

training_images = training_images / 255.0
val_images = val_images / 255.0

model = tf.keras.models.Sequential(
    [tf.keras.layers.Flatten(input_shape=(28,28)),
     tf.keras.layers.Dense(20, activation=tf.nn.relu),
     tf.keras.layers.Dense(10, activation=tf.nn.softmax)])
```









```
import tensorflow as tf

data = tf.keras.datasets.mnist
(training_images, training_labels), (val_images, val_labels) = data.load_data()

training_images = training_images / 255.0
val_images = val_images / 255.0

model = tf.keras.models.Sequential(
    [tf.keras.layers.Flatten(input_shape=(28,28)),
      tf.keras.layers.Dense(20, activation=tf.nn.relu),
      tf.keras.layers.Dense(10, activation=tf.nn.softmax)])
```

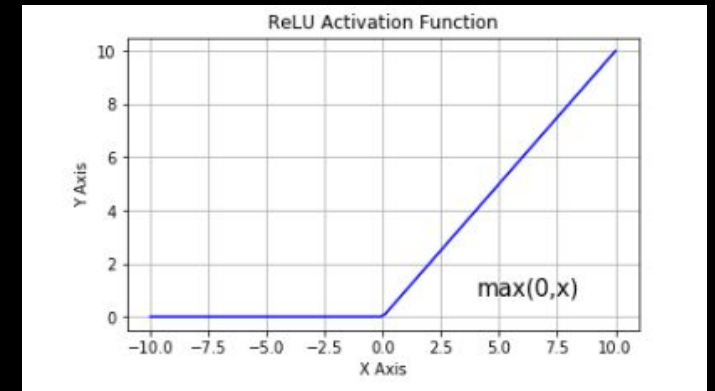


```
import tensorflow as tf

data = tf.keras.datasets.mnist
(training_images, training_labels), (val_images, val_labels) = data.load_data()

training_images = training_images / 255.0
val_images = val_images / 255.0

model = tf.keras.models.Sequential(
    [tf.keras.layers.Flatten(input_shape=(28,28)),
      tf.keras.layers.Dense(20, activation=tf.nn.relu),
      tf.keras.layers.Dense(10, activation=tf.nn.softmax)])
```



ReLU applies much-needed non-linearity into the model. Non-linearity is necessary to produce non-linear decision boundaries, so that the output cannot be written as a linear combination of the inputs.

```
import tensorflow as tf

data = tf.keras.datasets.mnist
(training_images, training_labels), (val_images, val_labels) = data.load_data()

training_images = training_images / 255.0
val_images = val_images / 255.0

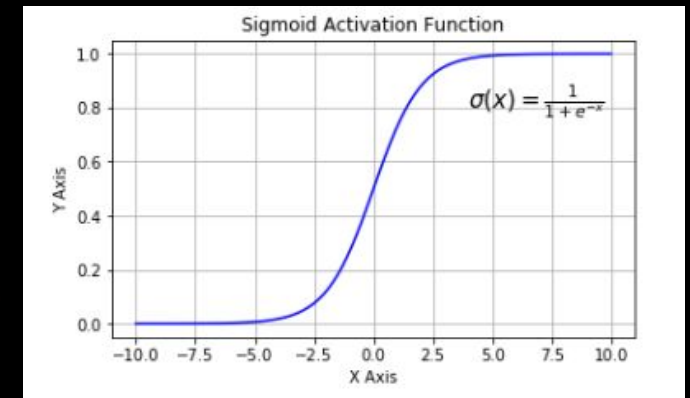
model = tf.keras.models.Sequential(
    [tf.keras.layers.Flatten(input_shape=(28,28)),
     tf.keras.layers.Dense(20, activation=tf.nn.relu),
     tf.keras.layers.Dense(10, activation=tf.nn.softmax)])
```

```
import tensorflow as tf

data = tf.keras.datasets.mnist
(training_images, training_labels), (val_images, val_labels) = data.load_data()

training_images = training_images / 255.0
val_images = val_images / 255.0

model = tf.keras.models.Sequential(
    [tf.keras.layers.Flatten(input_shape=(28,28)),
     tf.keras.layers.Dense(20, activation=tf.nn.relu),
     tf.keras.layers.Dense(10, activation=tf.nn.softmax)])
```



**SOFTMAX:** Generalization of the logistic function (or Sigmoid) to multiple dimensions. A softmax operation serves a key purpose: making sure the Neural Network (in this case, a DNN) outputs sum to 1. Because of this, softmax operations are useful to scale model outputs into probabilities.

```
model.compile(optimizer='adam',  
              loss='sparse_categorical_crossentropy',  
              metrics=['accuracy'])
```

```
model.compile(optimizer='adam',  
              loss='sparse_categorical_crossentropy',  
              metrics=['accuracy'])
```

### Mean Squared Error

$$MSE = \frac{1}{N} \sum (t_i - s_i)^2$$

Prediction  $s_i$

Ground Truth  $t_i$

### Cross Entropy Loss

$$CE = - \sum_i^C t_i \log(s_i)$$

Classes  $C$

Prediction  $s_i$

Ground Truth  $t_i \{0,1\}$

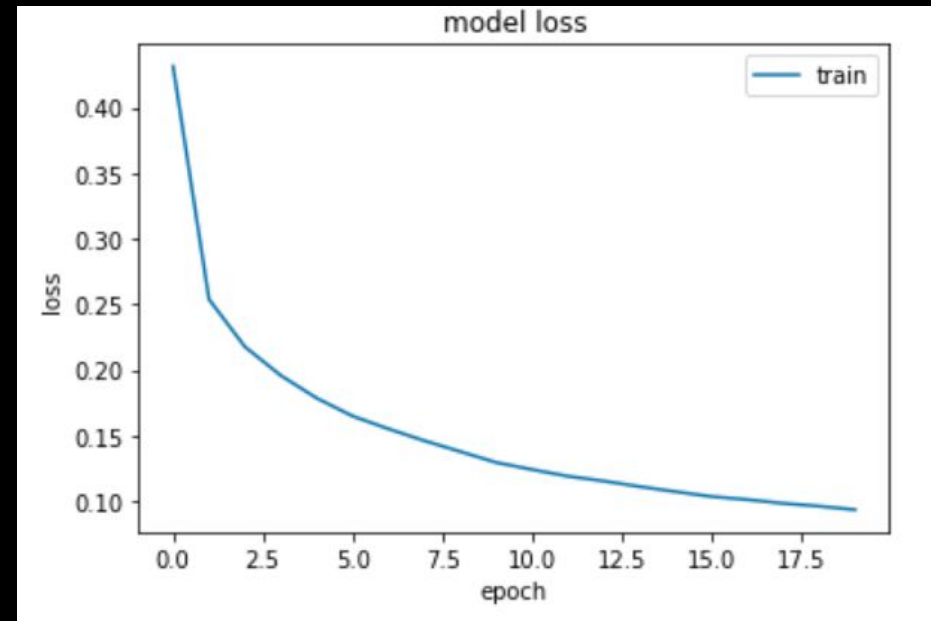
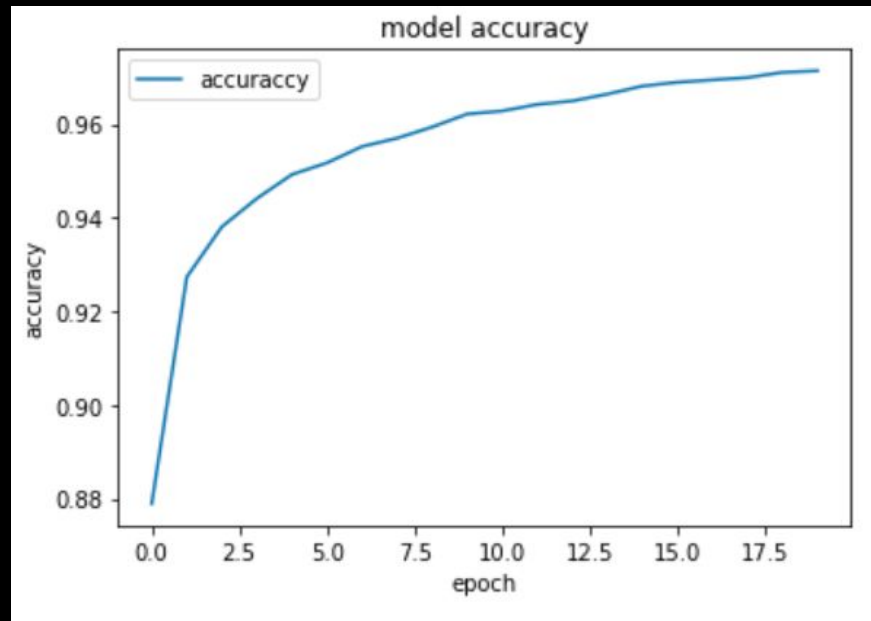


```
model.compile(optimizer='adam',  
              loss='sparse_categorical_crossentropy',  
              metrics=['accuracy'])
```

```
model.fit(training_images, training_labels, epochs=20)
```

```
model.compile(optimizer='adam',  
              loss='sparse_categorical_crossentropy',  
              metrics=['accuracy'])
```

```
model.fit(training_images, training_labels, epochs=20)
```



Evaluate  
Optimize



```
classifications = model.predict(val_images)
print(classifications[0])
print(test_labels[0])
```

```
[2.4921512e-09 1.3765138e-10 8.8281205e-08
1.0477231e-03 2.8455029e-12 4.0820678e-06
2.0070659e-16 9.9894780e-01 1.0296049e-07
2.9972372e-07]
```

7

# Digits Classification using DNN with TF2

## Code Time!

TF\_MNIST\_Classification.ipynb



# Going deeper with Deep Learning

Initializing neural networks

<https://www.deeplearning.ai/ai-notes/initialization/>

Neural networks – PlayList - 3Blue1Brown

[https://www.youtube.com/playlist?list=PLZHQObOWTQDNU6R1\\_67000Dx\\_ZCJB-3pi](https://www.youtube.com/playlist?list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi)

An introductory lecture for MIT course 6.S094 by Prof. Lex Fridman

<https://youtu.be/O5xeyoRL95U>

A Complete Machine Learning Package by Jean de Dieu Nyandwi

[https://github.com/Nyandwi/machine\\_learning\\_complete](https://github.com/Nyandwi/machine_learning_complete)

# Reading Material

# Main references

- [Harvard School of Engineering and Applied Sciences - CS249r: Tiny Machine Learning](#)
- [Professional Certificate in Tiny Machine Learning \(TinyML\) – edX/Harvard](#)
- [Introduction to Embedded Machine Learning - Coursera/Edge Impulse](#)
- [Computer Vision with Embedded Machine Learning - Coursera/Edge Impulse](#)
- Fundamentals textbook: [“Deep Learning with Python” by François Chollet](#)
- Applications & Deploy textbook: [“TinyML” by Pete Warden, Daniel Situnayake](#)
- Deploy textbook [“TinyML Cookbook” by Gian Marco Iodice](#)

I want to thank **Shawn Hymel** and Edge Impulse, **Pete Warden** and **Laurence Moroney** from Google, Professor **Vijay Janapa Reddi** and **Brian Plancher** from Harvard, and the rest of the **TinyMLedu** team for preparing the excellent material on TinyML that is the basis of this course at UNIFEI.

The IESTI01 course is part of the **TinyML4D**, an initiative to make TinyML education available to everyone globally.

Thanks



**UNIFEI**