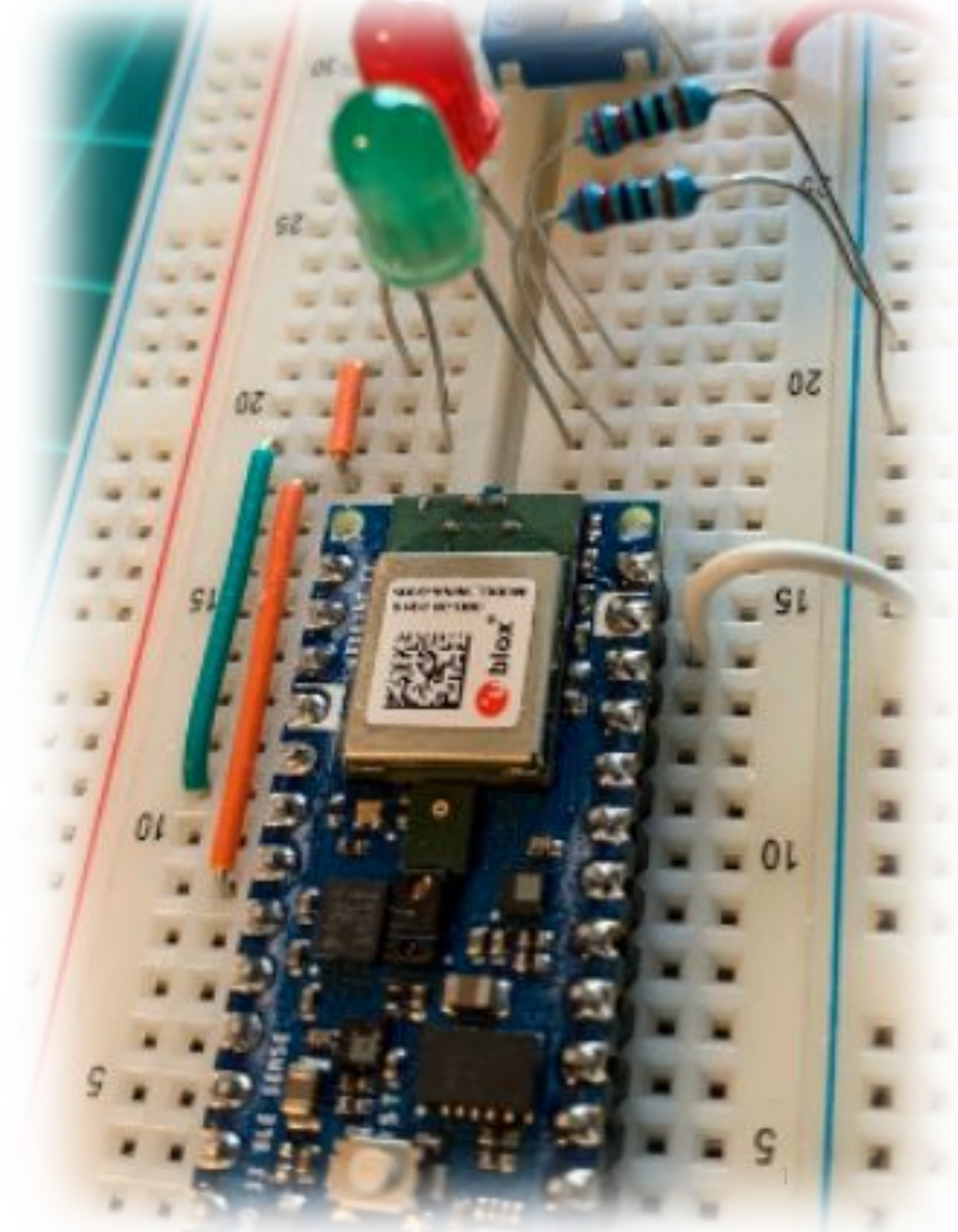
IESTI01 – TinyML

Embedded Machine Learning

9. The Building Blocks of Deep Learning – Part C
   - DNN Recap & ML Metrics

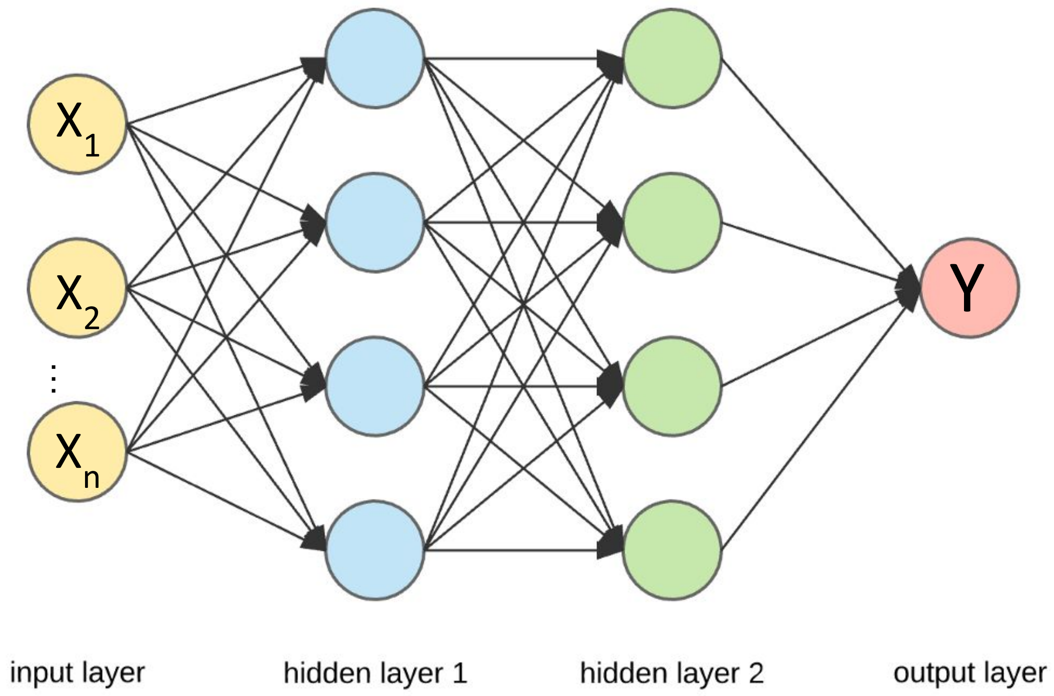
Prof. Marcelo Rovai
UNIFEI

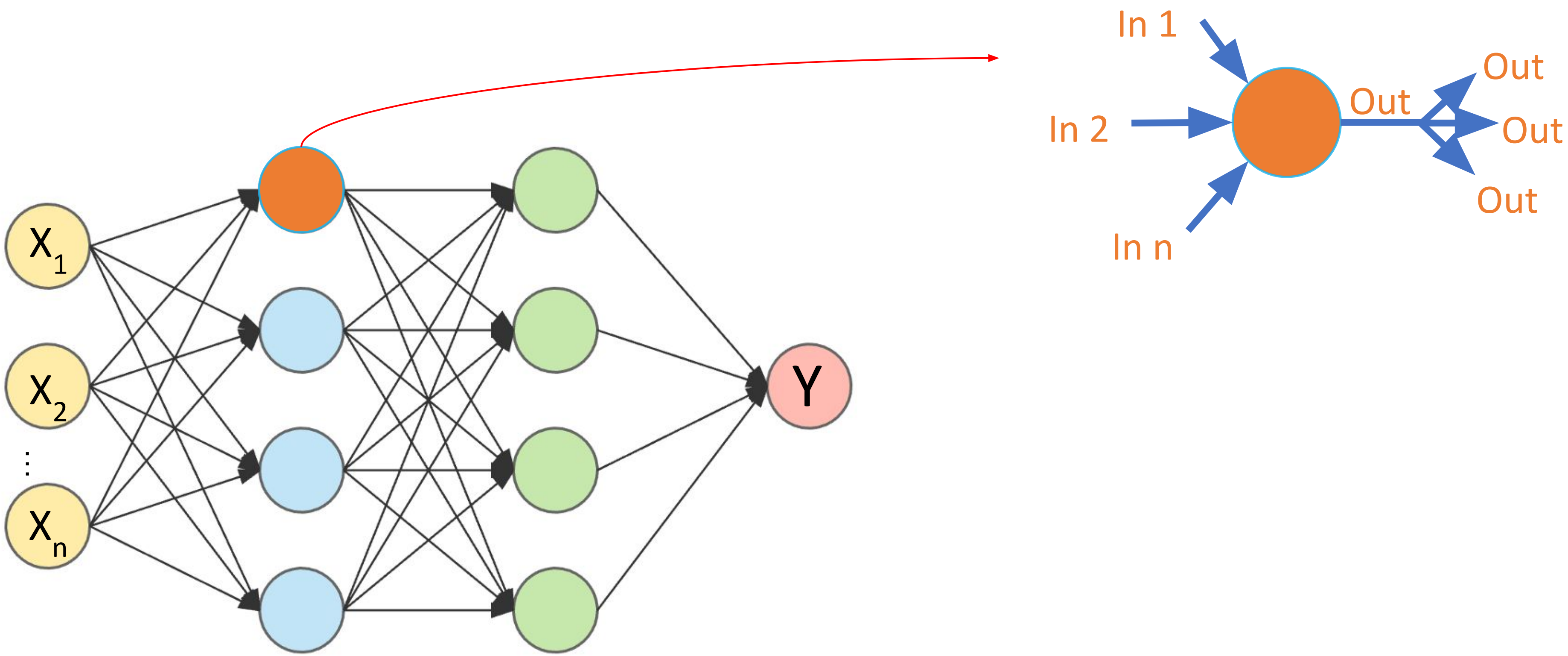


DNN Dense Neural Network

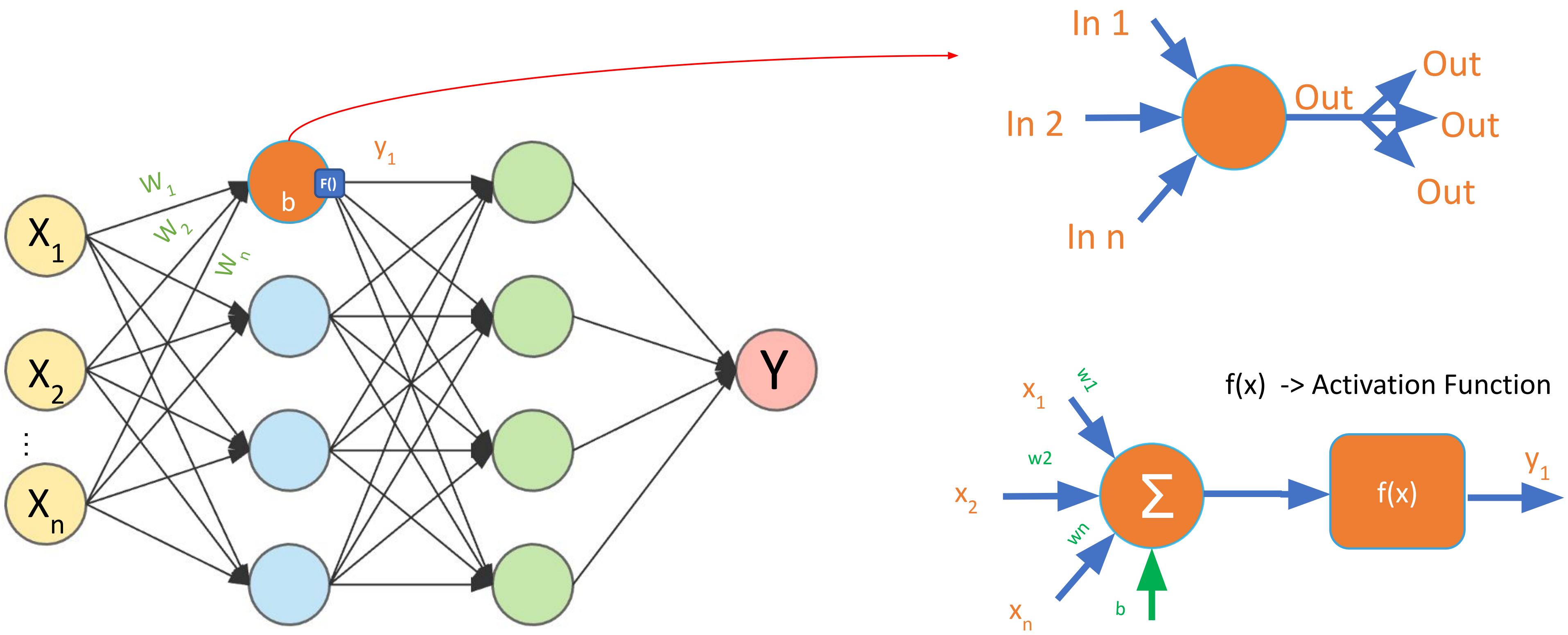
Recap

Supervised Machine Learning with DNN



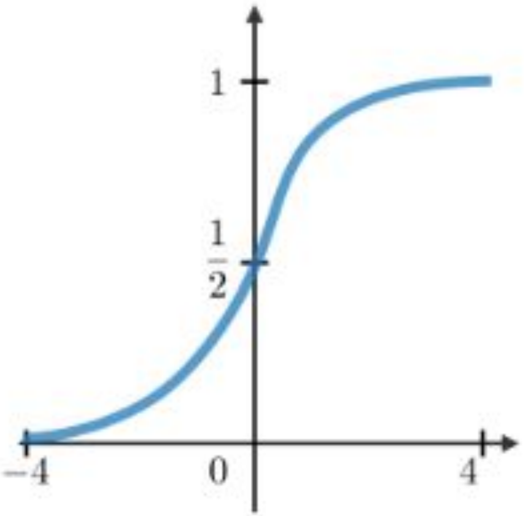
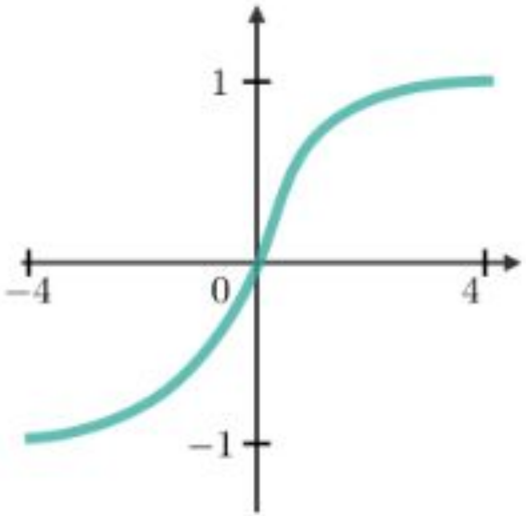
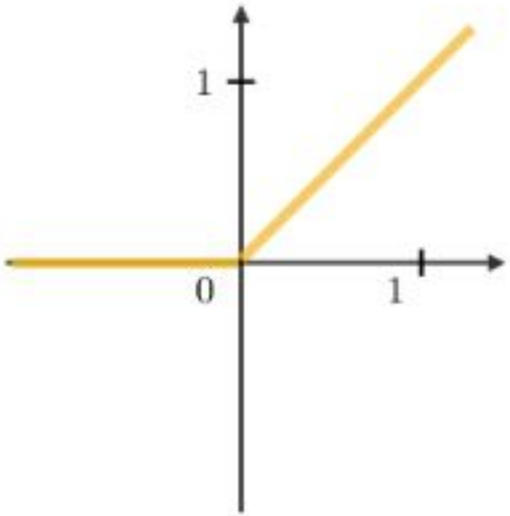


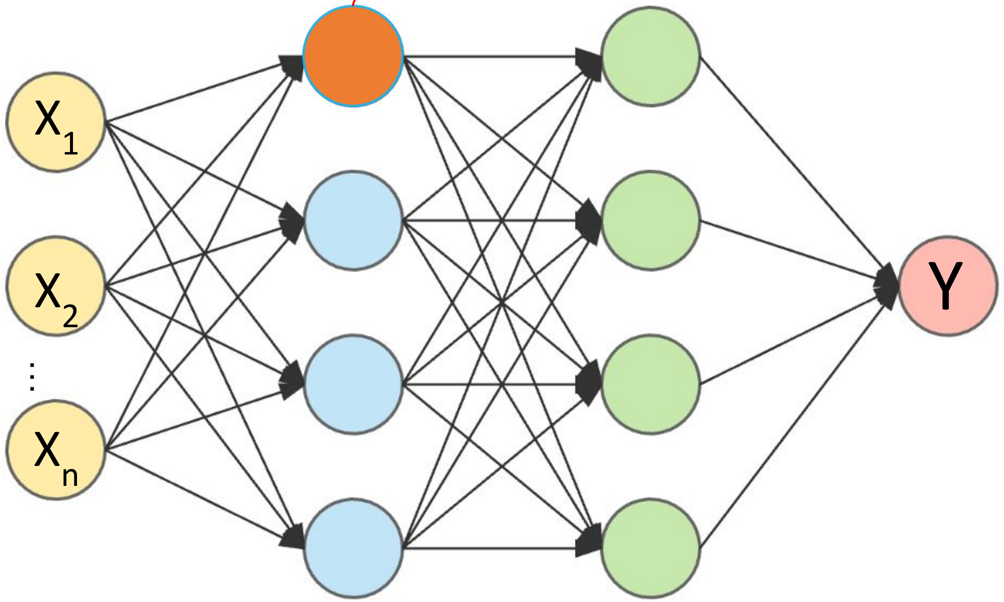
X1
X2
Xn
Y
In 1
In 2
In n
Out
Out
Out
Out

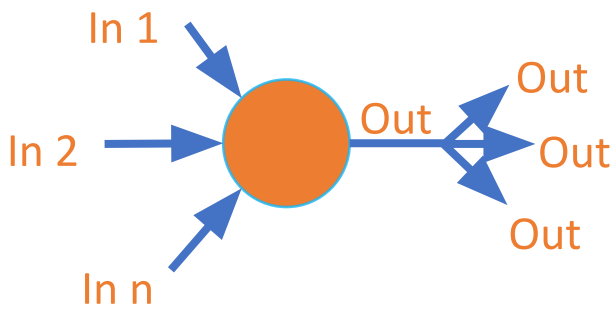


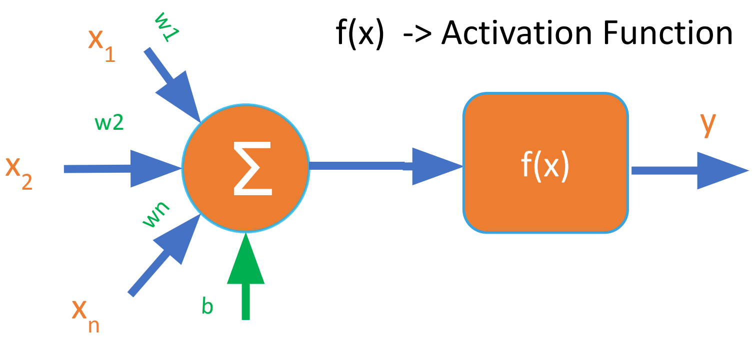
$$y = f(\sum_{i=1}^{n} x_i w_i + b)$$

Activation Functions

Sigmoid	Tanh	RELU
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$
		

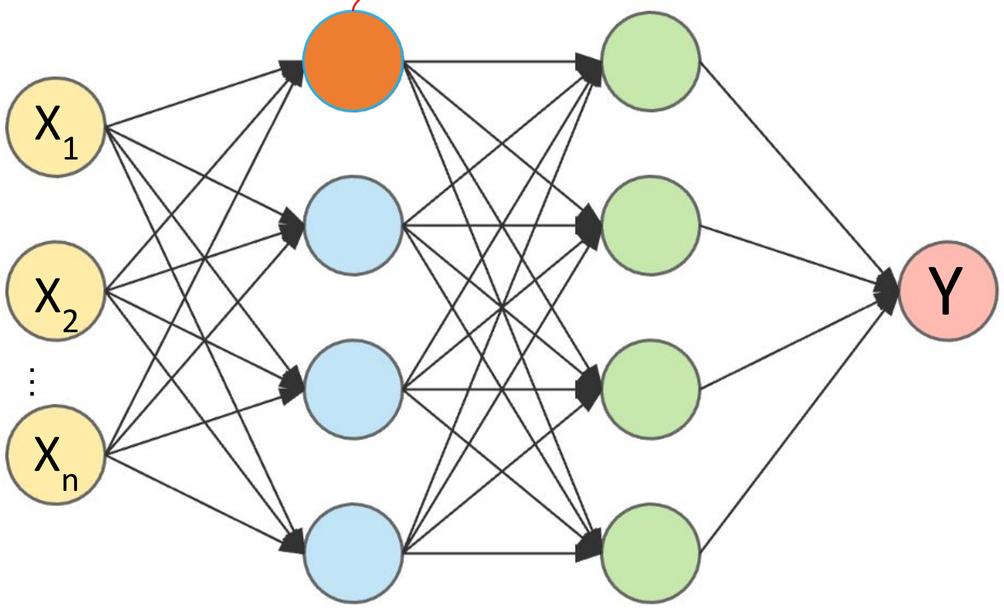


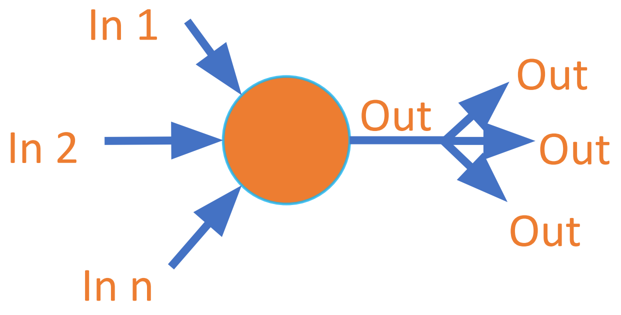


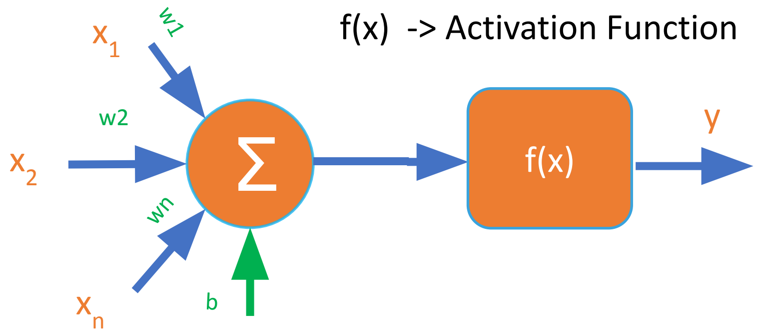


Parameters to be found during training, to reach minimum error

$$y = f(\sum_{i=1}^{n} x_i w_i + b)$$



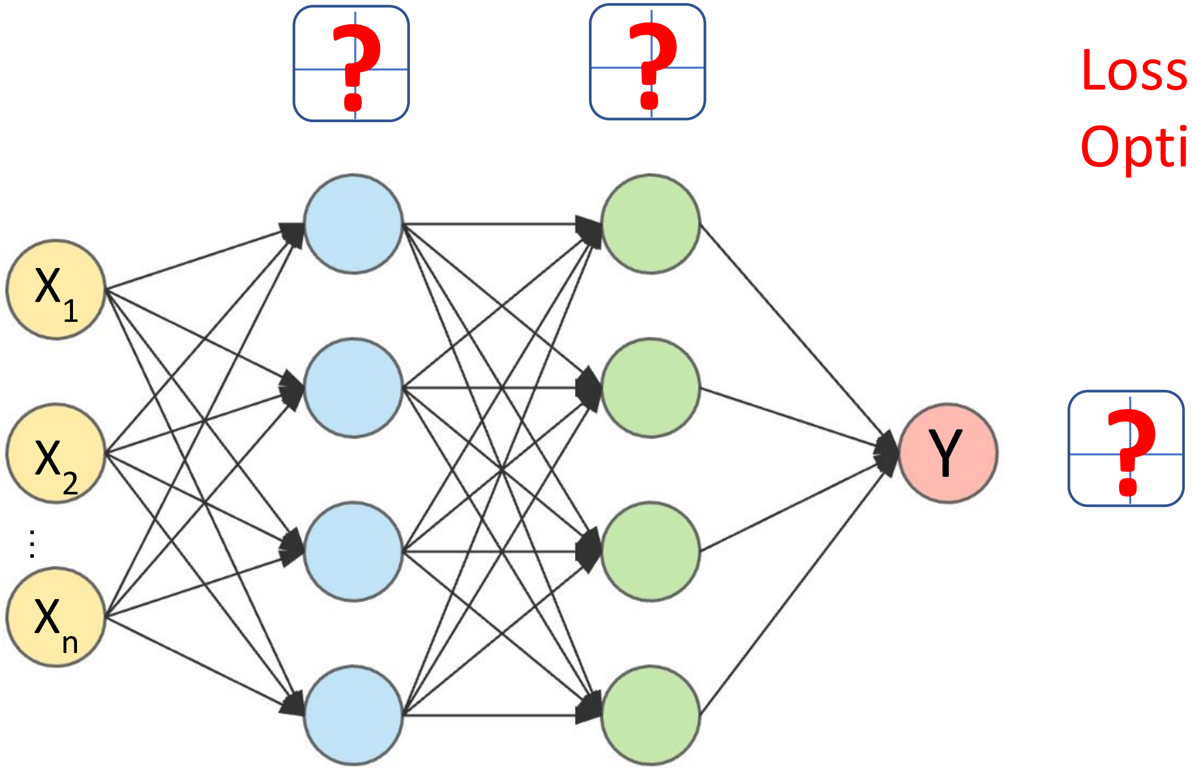




- Error Measurement (Loss)
- Optimization

Parameters to be found during training, to reach minimum error

$$y = f(\sum_{i=1}^{n} x_i w_i + b)$$

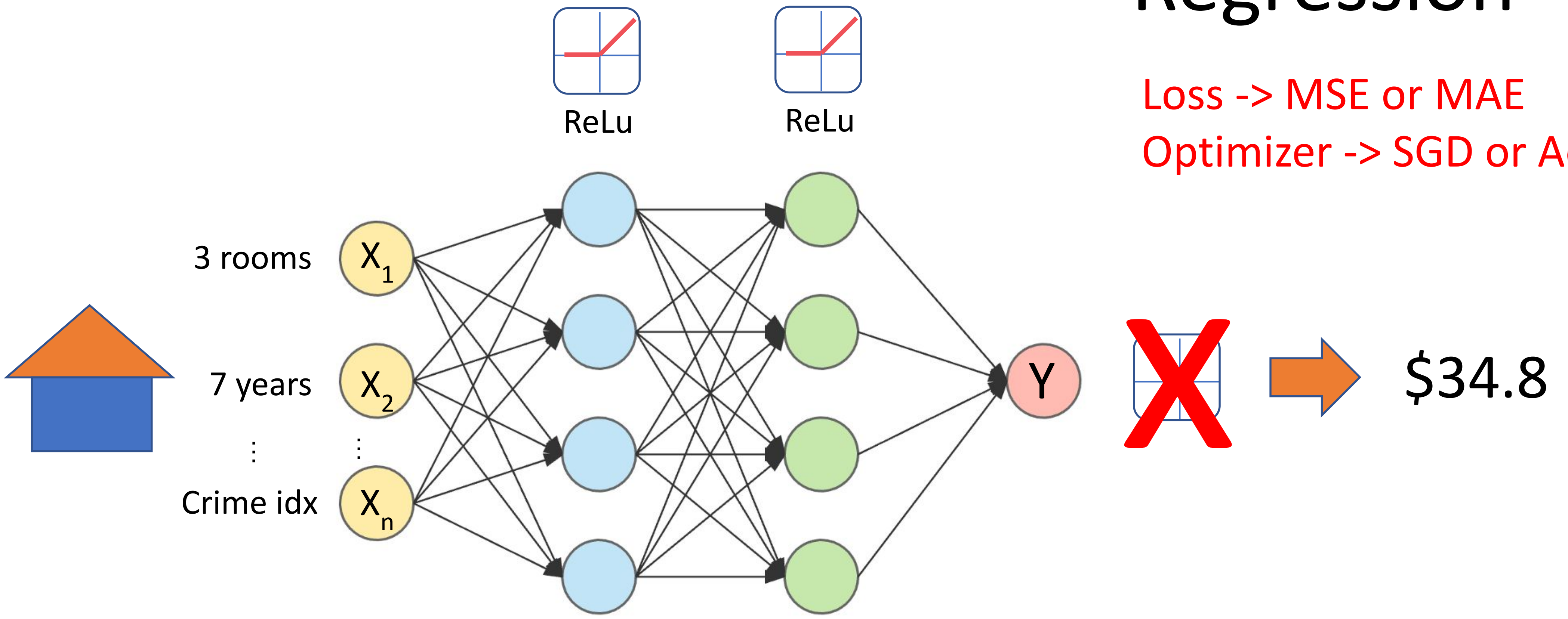


Loss -> ?

Optimizer -> ?

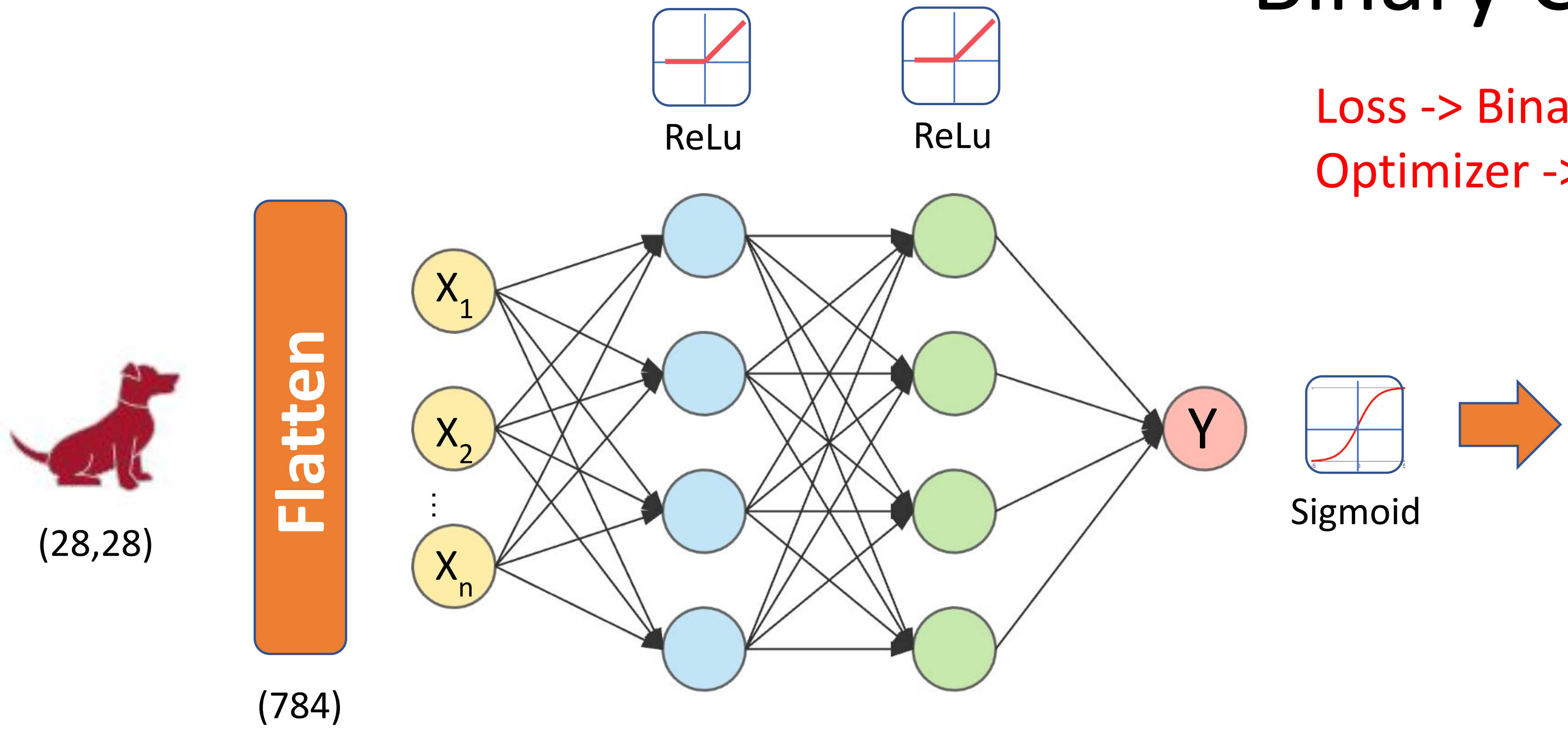
Regression

Loss -> MSE or MAE
Optimizer -> SGD or Adam



Binary Classification

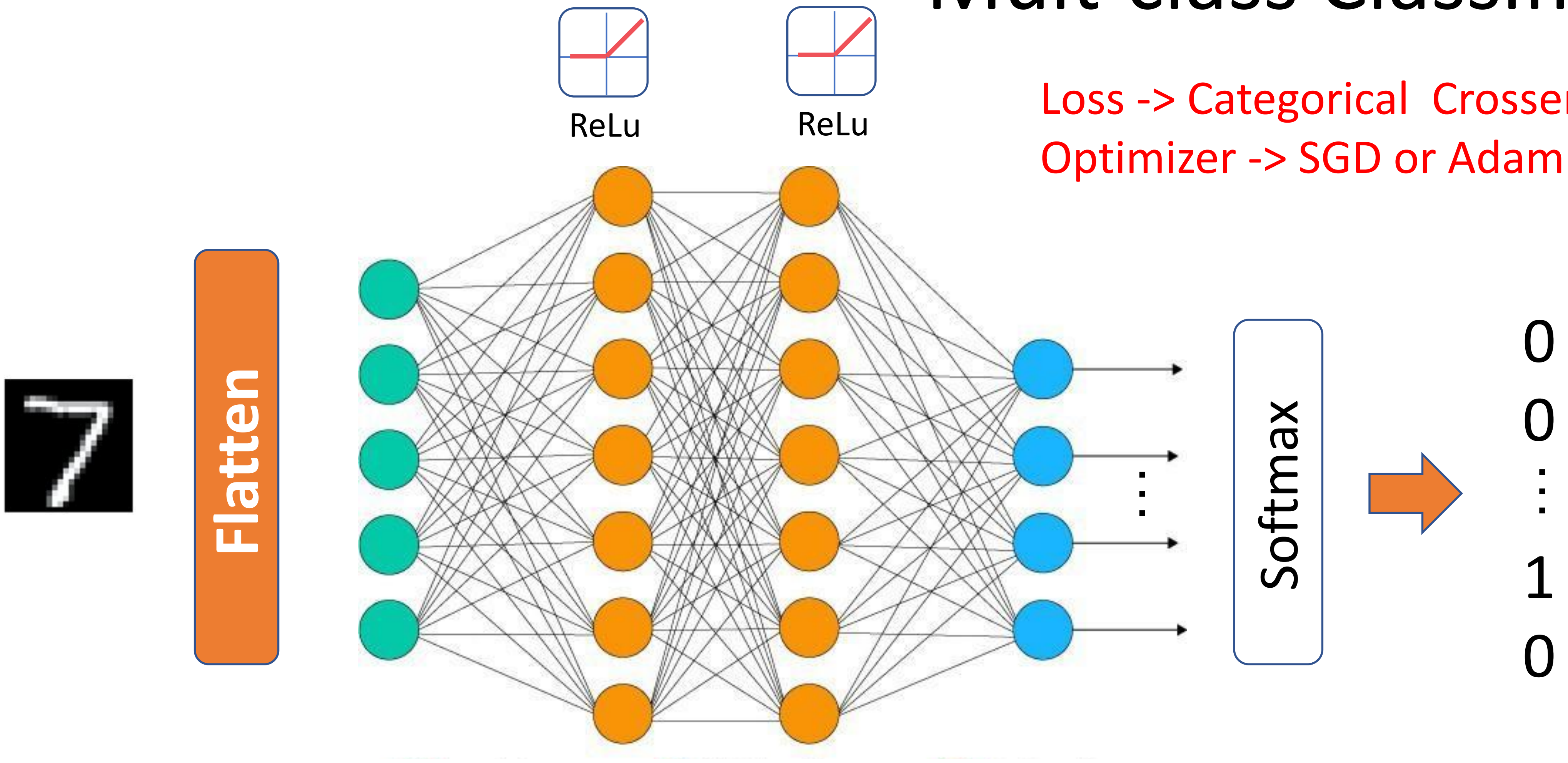
Loss -> Binary Crossentropy
Optimizer -> SGD or Adam



0: Cat
1: Dog

Mult-class Classification

Loss -> Categorical Crossentropy *
Optimizer -> SGD or Adam



* or “Sparse Categorical Crossentropy” if label is 1, 2, 3, …

Going Further

The Datasets to training and test




Classifying Shoes

Steps to take

1. Get as many examples of shoes as possible
2. Train using these examples
3. Profit!


FINLAND
na
višjem

Steps to take

1. Get as many examples of shoes as possible
2. Train using these examples
3. Profit!

```
Training accuracy: .920
Training accuracy: .935
Training accuracy: .947
Training accuracy: .961
Training accuracy: .977
Training accuracy: .995
Training accuracy: 1.00
```

Steps to take

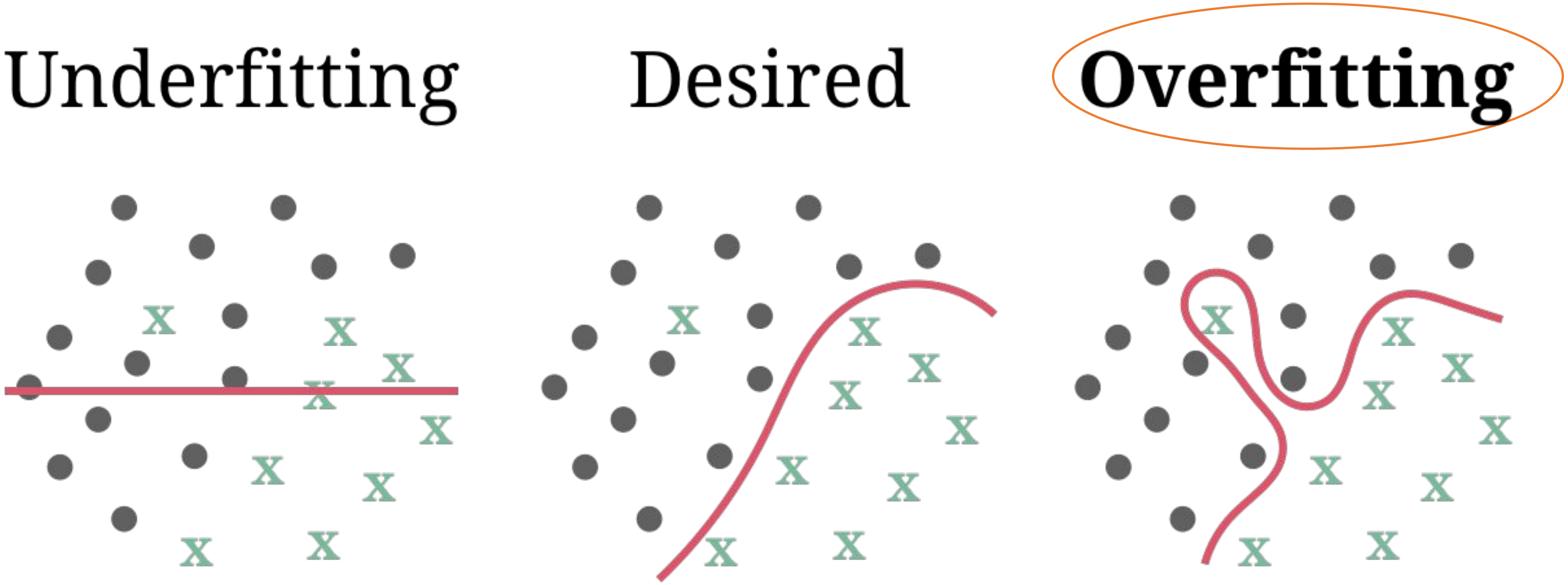
1. ~~Get as many examples of shoes as possible~~
2. ~~Train using these examples~~
3. Profit?

Data

The network ‘sees’ everything. Has no context for measuring how well it does with data it has never previously been exposed to.

Data

The network 'sees' everything. Has no context for measuring how well it does with data it has never previously been exposed to.



Data	Validation Data

The network ‘sees’ a subset of your data. You can use the rest to measure its performance against previously unseen data.

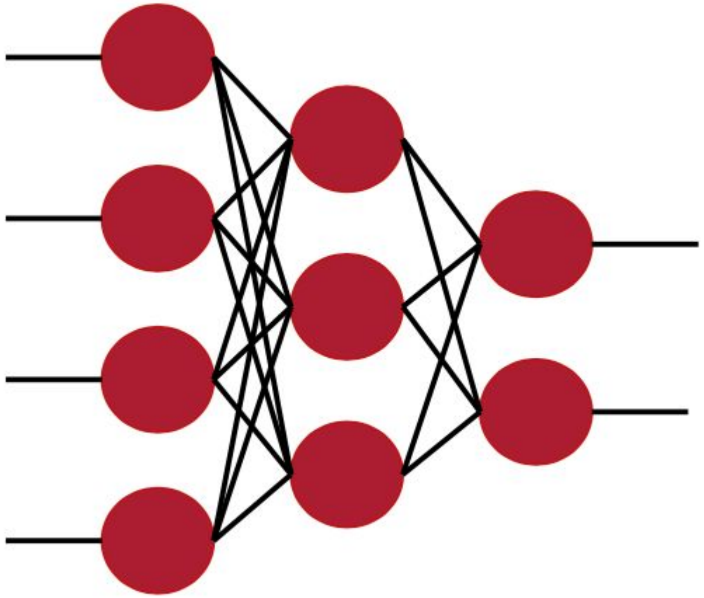
Data	Validation Data	Test Data

The network 'sees' a subset of your data. You can use an unseen subset to measure its accuracy while training (validation), and then another subset to measure its accuracy after it's finished training (test).

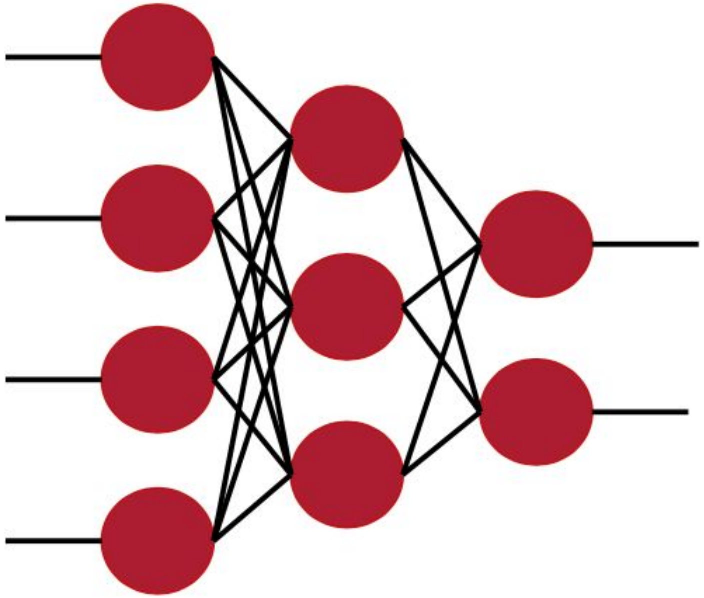
Is used to evaluate the current training epoch

Is used to evaluate the final model after training

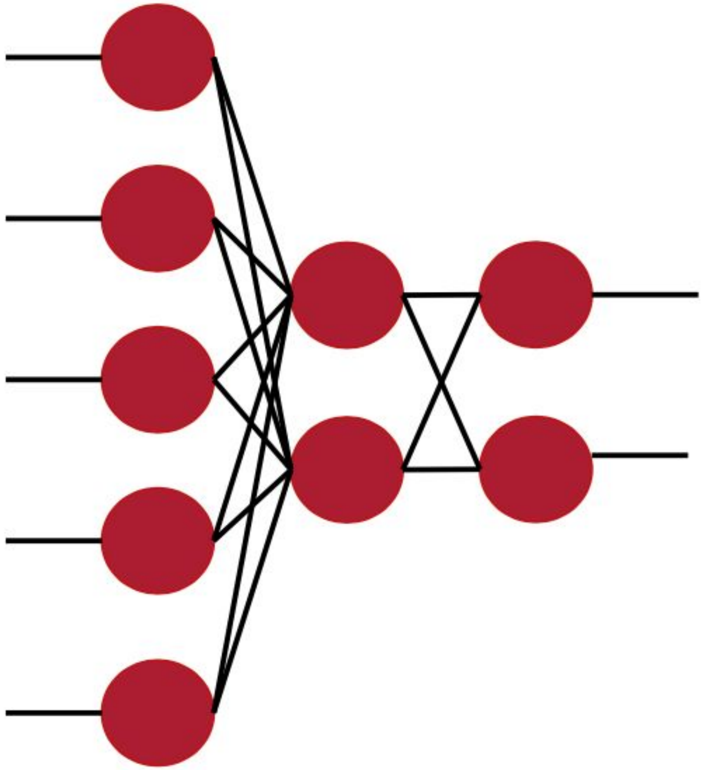
Data	Validation Data	Test Data
Accuracy: 0.999	Accuracy: 0.920	Accuracy: 0.800



Data	Validation Data	Test Data
Accuracy: 0.999	Accuracy: 0.920	Accuracy: 0.800



Data	Validation Data	Test Data
Accuracy: 0.942	Accuracy: 0.930	Accuracy: 0.925



Digits Classification: validation and test dataset

Code Time!

[TF_MNIST_Classification_v2.ipynb]

```python
data = tf.keras.datasets.mnist

(tt_images, tt_labels), (test_images, test_labels) = data.load_data()
```

```python
print(tt_images.shape)
print(tt_labels.shape)
```

```
(60000, 28, 28)
(60000,)
```

```python
print(test_images.shape)
print(test_labels.shape)
```

```
(10000, 28, 28)
(10000,)
```

```
val_images = tt_images[:10000]
val_labels = tt_labels[:10000]
```

```
train_images = tt_images[10000:]
train_labels = tt_labels[10000:]
```

Split tt data in:

- train (50,000) and,
- validation (10,000)

```
print(train_images.shape)
print(train_labels.shape)
```

```
(50000, 28, 28)
(50000,)
```

```
print(val_images.shape)
print(val_labels.shape)
```

```
(10000, 28, 28)
(10000,)
```

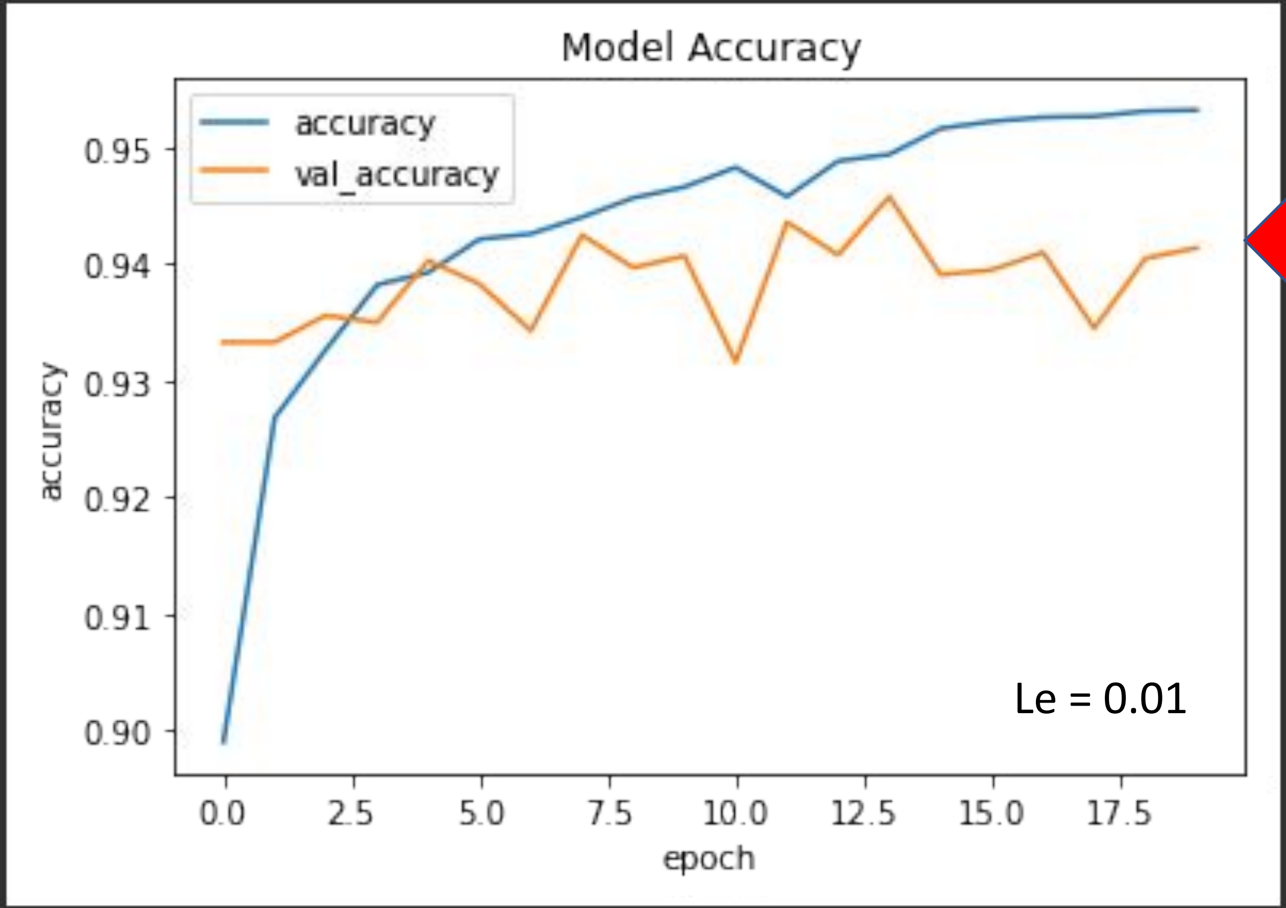
```
history = model.fit(
    train_images,
    train_labels,
    epochs=20,
    validation_data=(val_images, val_labels)
    )
```

You could leave the training data with all samples, and alternatively use:

- *validation_split=0.1* instead of *validation_data=(val_images, val_labels)*.

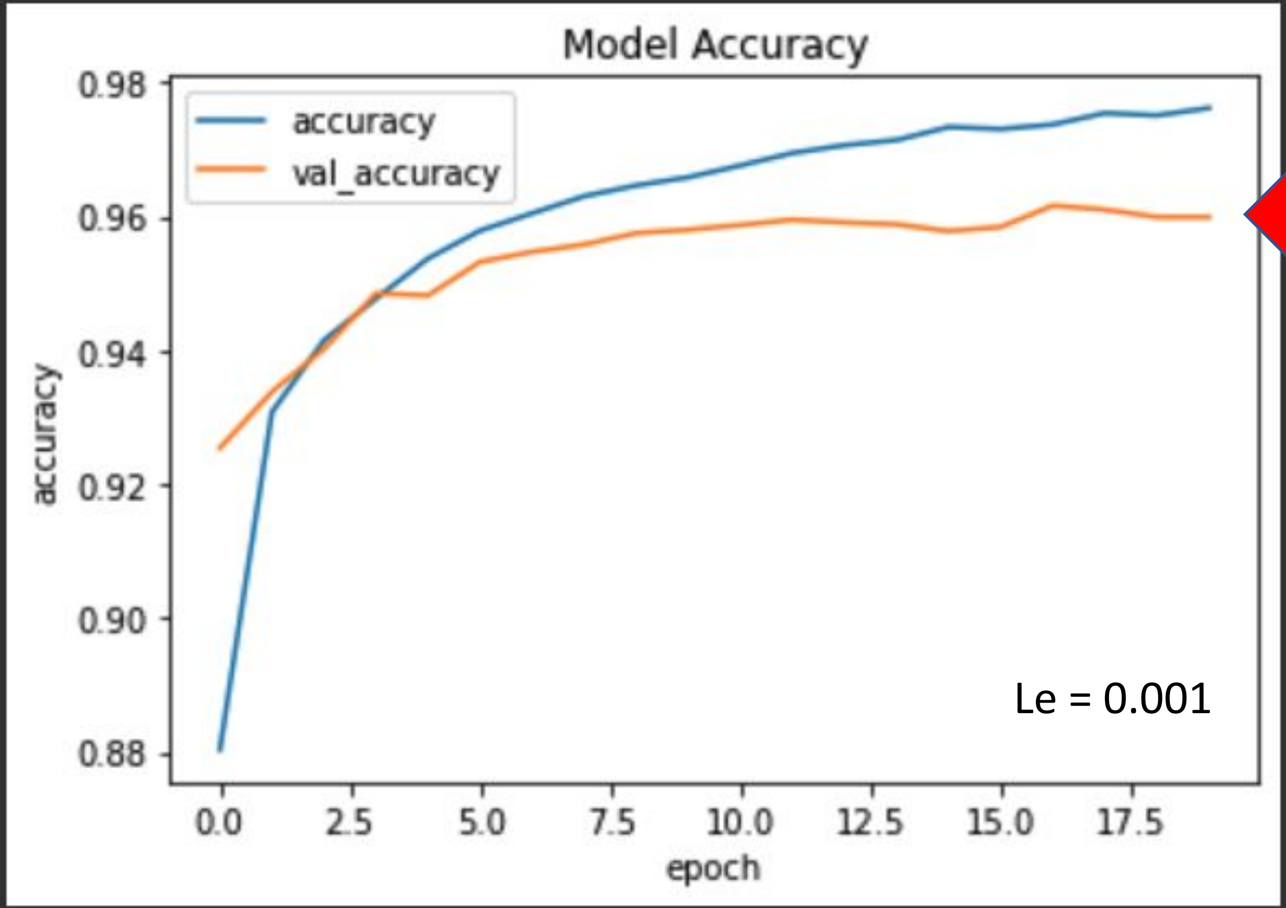
In this case, TF will split the validation data by itself.

```python
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.title('Model Accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(loc='upper left')
plt.show()
```



If validation accuracy seems "unstable", could be that Learning Rate is high (try to reduce it).

```python
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.title('Model Accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(loc='upper left')
plt.show()
```



If validation accuracy goes down (or became stable) , even if train accuracy goes up, means that probably the model is overfitting. In this case the training (epochs) should terminate.

```
model.evaluate(test_images, test_labels)
```

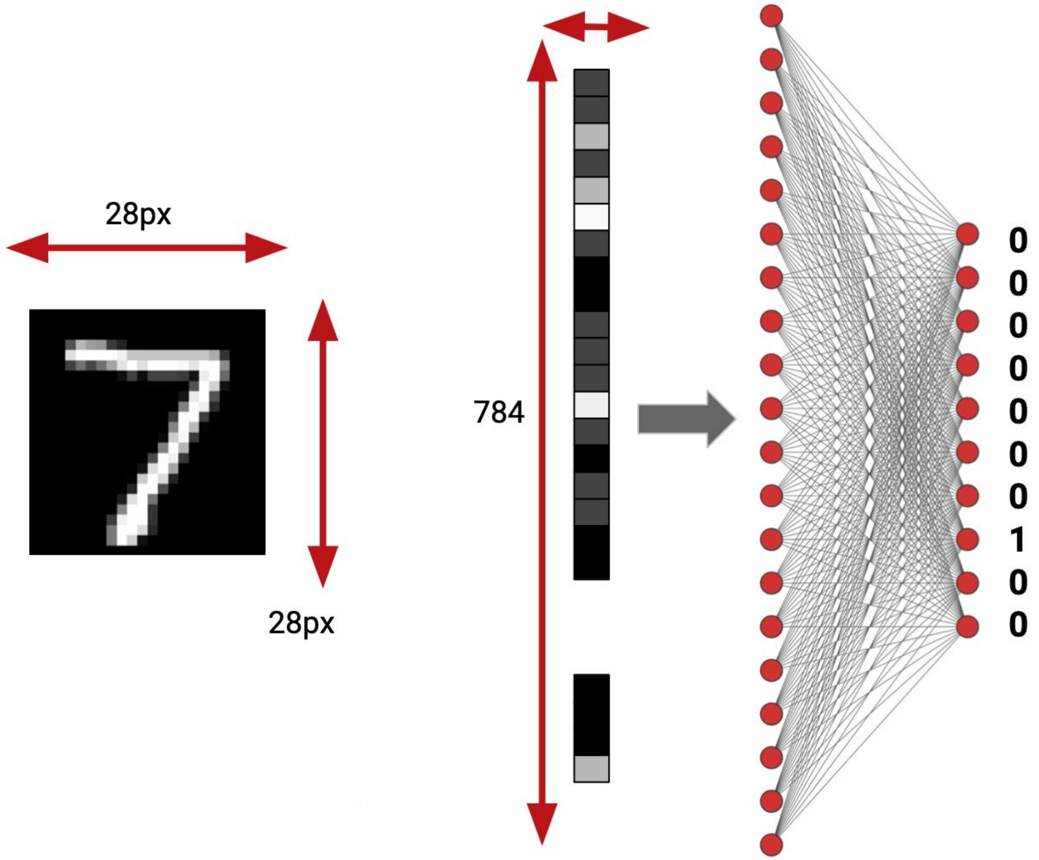
```
313/313 [==============================] - 1s 2ms/step - loss: 0.1495 - accuracy: 0.9569
```

Data | Validation Data | Test Data

Accuracy: 0.976

Accuracy: 0.963

Accuracy: 0.957



In summary

Training Data -> Used to train model parameters

Validation Data -> Used to determine what model hyperparameters to adjust (and re-training)

Test Data -> Used to get model final performance metric

Going Further

Classification Model Performance Metrics

Class = [1]

actual = [1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0]

prediction = [0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1]

Class = [0]

Model Performance (Confusion Matrix)

		predicted condition	
	12 pictures, 8 of cats and 4 of dogs	**Cat** [1]	**Dog** [0]
true condition	**Cat** [1]	**6**	**2**
	Dog [0]	**1**	**3**

Model Performance (Confusion Matrix)

		predicted condition	
	12 pictures, 8 of cats and 4 of dogs	**Cat** [1]	**Dog** [0]
true condition	**Cat** [1]	True Positive (TP) **6**	False Negative (FN) (type II error) **2**
	Dog [0]	False Positive (FP) (Type I error) **1**	True Negative (TN) **3**

Model Performance (Confusion Matrix)

		predicted condition	
	total population (P + N)	prediction positive (PP)	prediction negative (PN)
true condition	condition positive (P)	**True Positive (TP)**	**False Negative (FN)** (type II error)
	condition negative (N)	**False Positive (FP)** (Type I error)	**True Negative (TN)**

https://en.wikipedia.org/wiki/Confusion_matrix

Type I error (false positive)

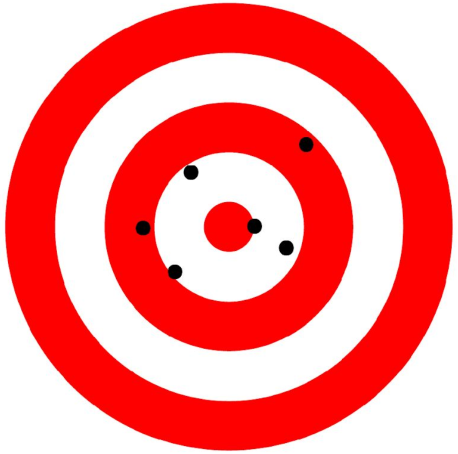
Type II error (false negative)

Precision vs. Accuracy

In a set of measurements:

- **Accuracy** is closeness of the measurements to a specific value
- **Precision** is the closeness of the measurements to each other.

High Precision, High Accuracy



Low Precision, High Accuracy

High Precision, Low Accuracy

Low Precision, Low Accuracy

Accuracy , Precision and Recall

$$\text{Accuracy} = \frac{TP + TN}{(P + N)} = \frac{TP + TN}{(TP + TN + FP + FN)} = \frac{6 + 3}{(6 + 3 + 1 + 2)} = \frac{9}{12} = 0.75$$

$$\text{Precision} = \frac{TP}{(TP + FP)} = \frac{6}{(6 + 1)} = \frac{6}{7} = 0.86$$

$$\frac{\text{Total Positive}}{\text{Total Predict Positive}}$$

$$\text{Recall} = \frac{TP}{(TP + FN)} = \frac{6}{(6 + 2)} = \frac{6}{8} = 0.75$$

(or Sensitivity)

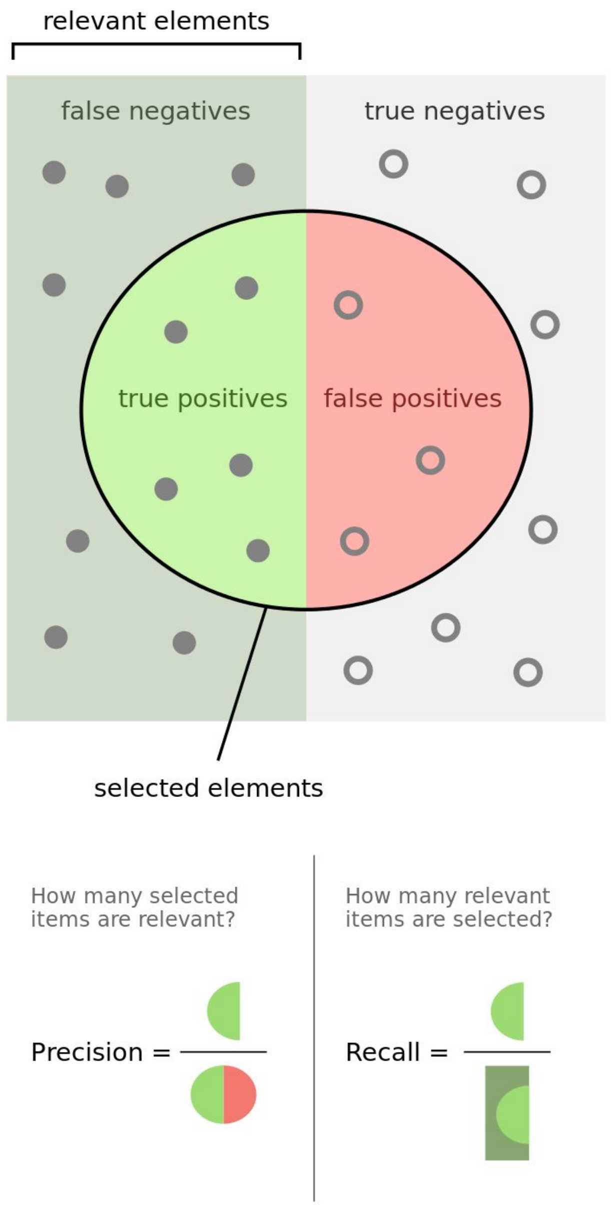
$$\frac{\text{Total Positive}}{\text{Total Actual Positive}}$$

F1-Score

$$F1 = 2 \times \frac{(\text{Precision} * \text{Recall})}{(\text{Precision} + \text{Recall})}$$

$$F1 = 2 \times \frac{(0.86 * 0.75)}{(0.86 + 0.75)} = 2 \times \frac{0.65}{1.61} = 0.80$$

The F1-score is a way of combining the precision and recall of the model



https://en.wikipedia.org/wiki/F-score#Formulation

Classification Report
Code Time!

Classification_Report.ipynb

```python
from sklearn.metrics import classification_report
```

```python
actual = [1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0]
prediction = [0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1]
```

```python
target_names = ['Dogs', 'Cats']
```

```python
print(classification_report(actual, prediction, target_names=target_names))
```

```
              precision    recall  f1-score   support

        Dogs       0.60      0.75      0.67         4
        Cats       0.86      0.75      0.80         8

    accuracy                           0.75        12
   macro avg       0.73      0.75      0.73        12
weighted avg       0.77      0.75      0.76        12
```

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