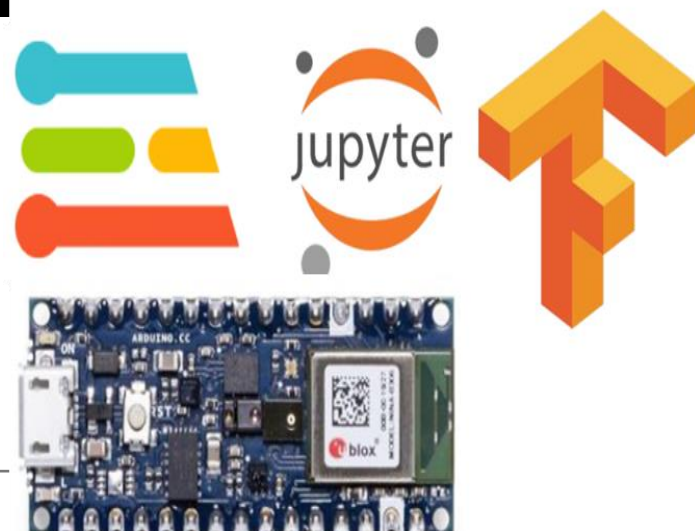




# Advanced Microprocessors

## DNN AND ML METRICS

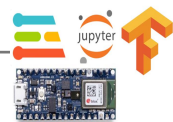
Dennis A. N. Gookyi





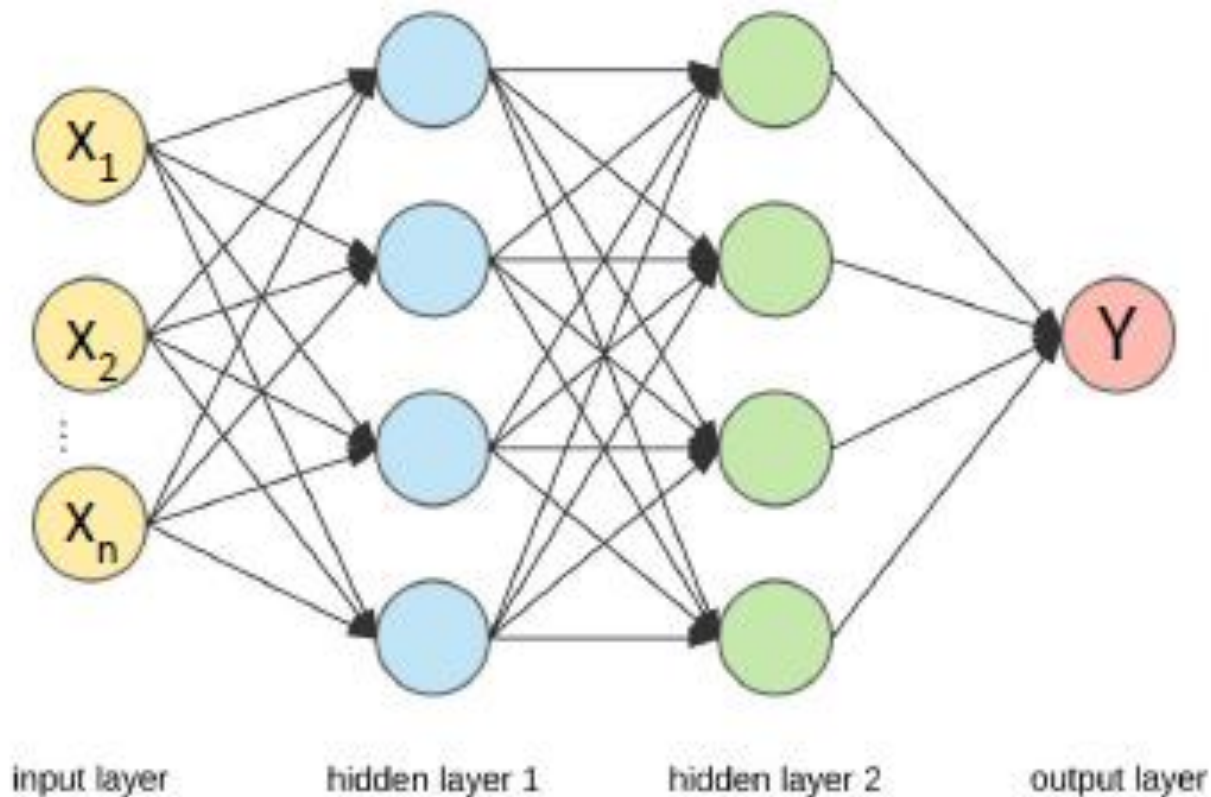
# CONTENTS

## ❖ DNN and ML Metrics



# DNN

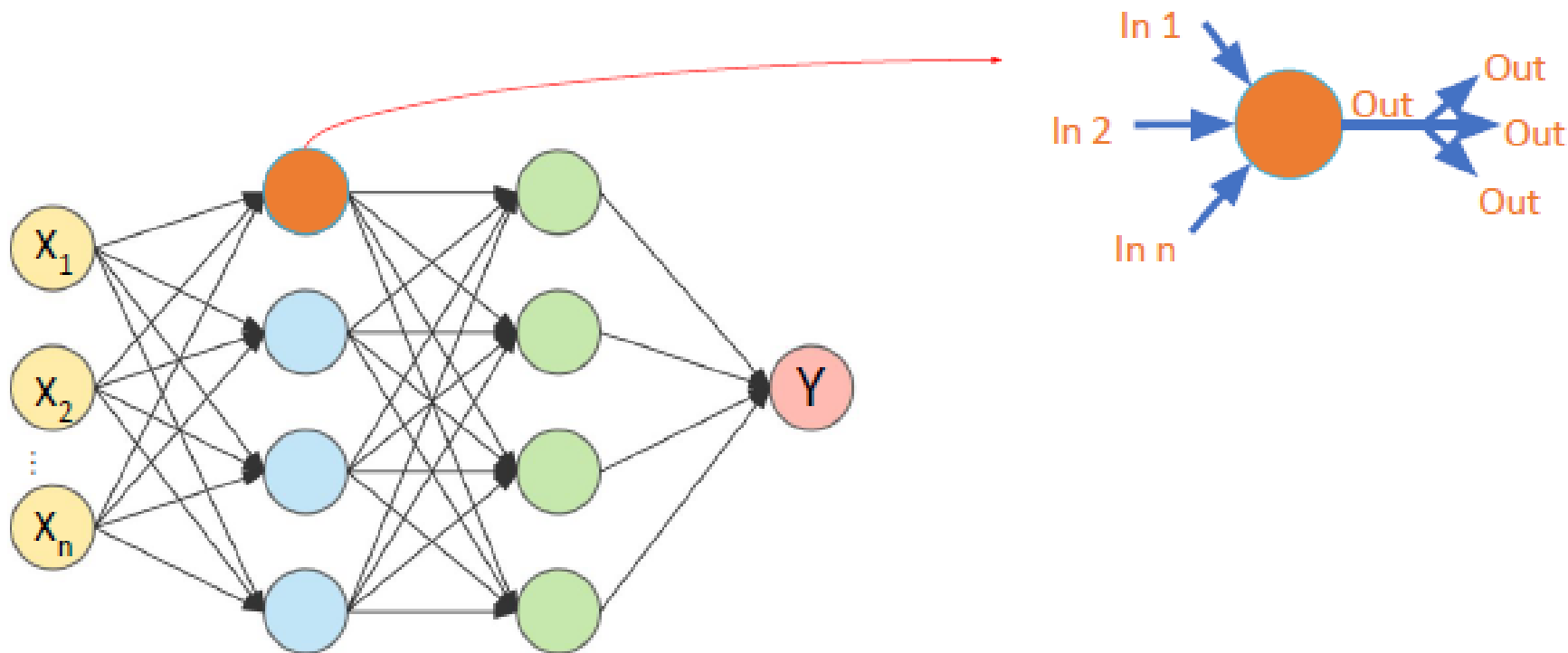
## ❖ Supervised machine learning with DNN





# DNN

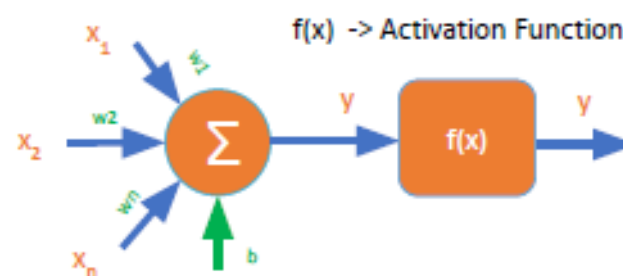
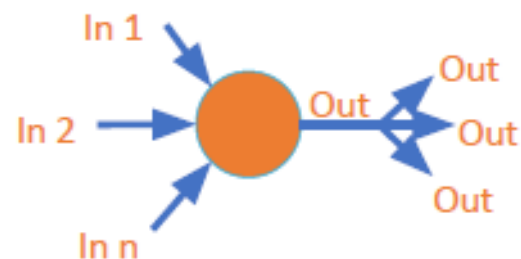
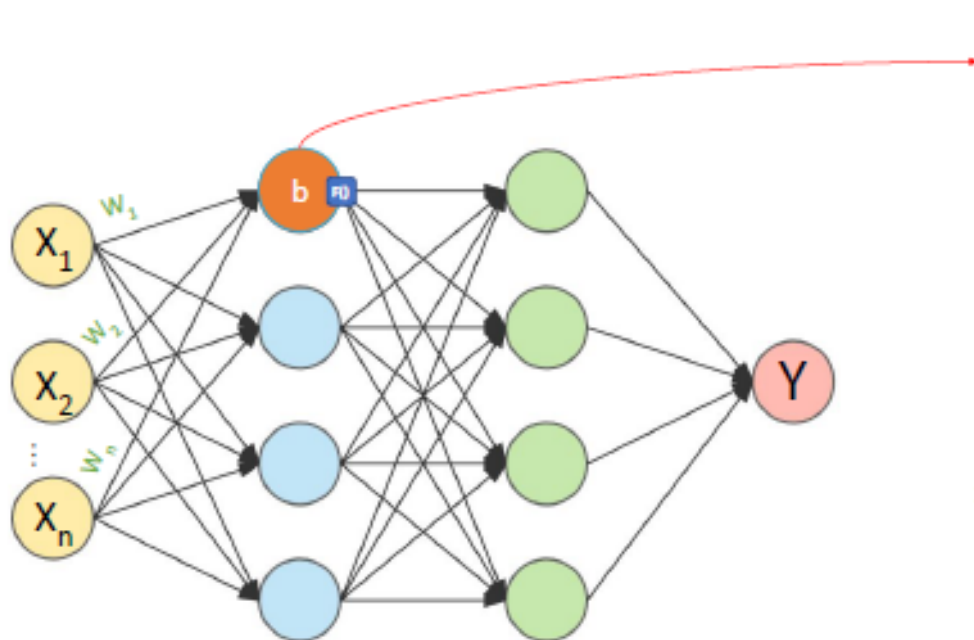
## ❖ Supervised machine learning with DNN





# DNN

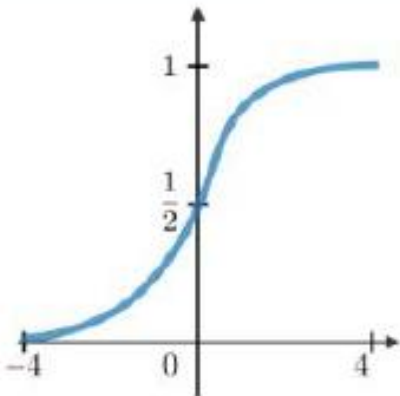
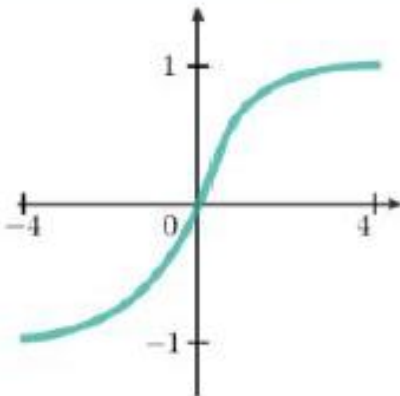
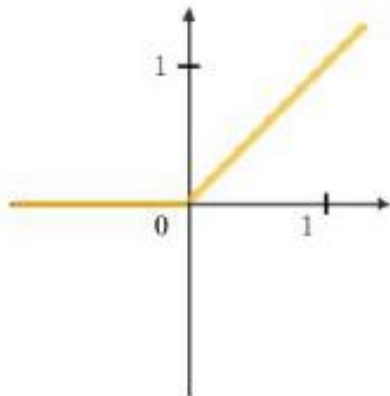
## ❖ Supervised machine learning with DNN

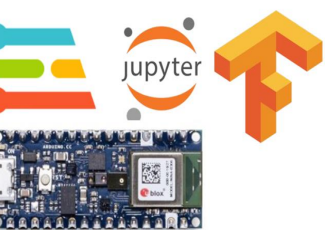


$$y = f\left(\sum_{i=1}^n x_i w_i + b\right)$$

## ❖ Supervised machine learning with DNN

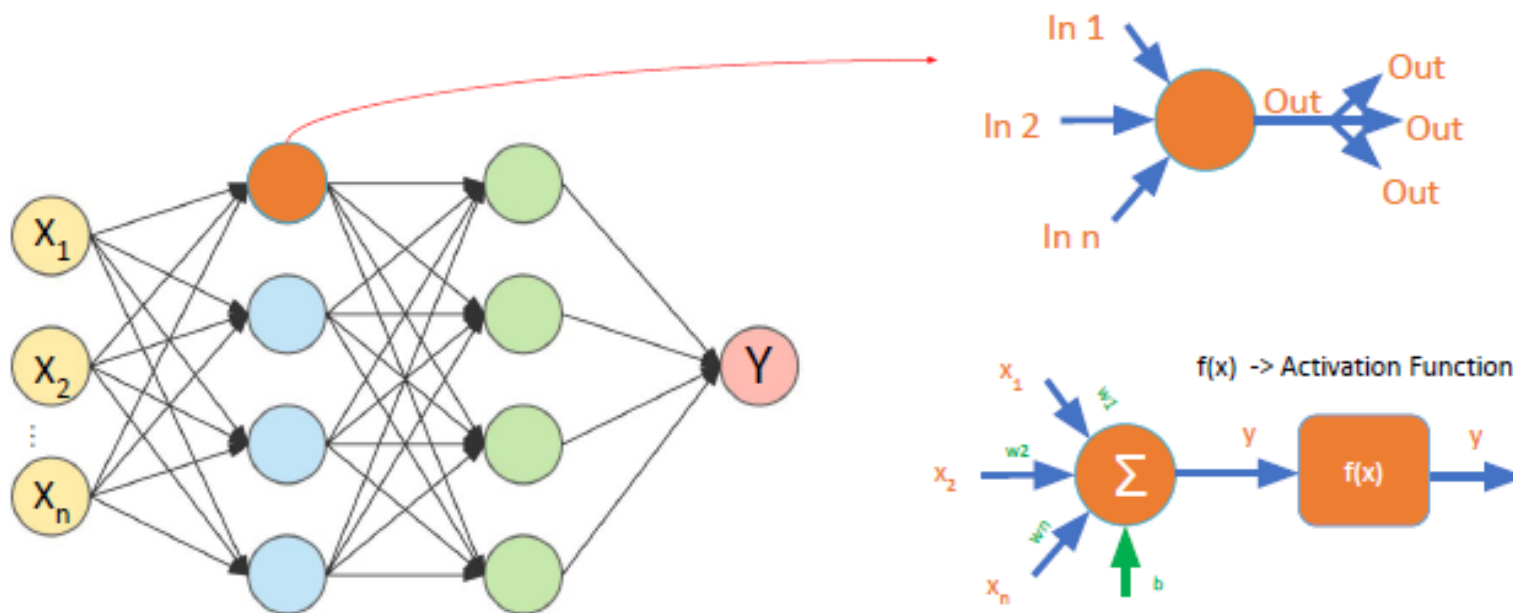
### □ Activation Functions

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



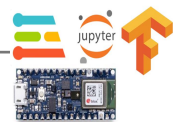
# DNN

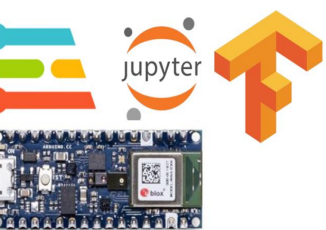
## ❖ Supervised machine learning with DNN



Parameters to be found during training, to reach minimum error

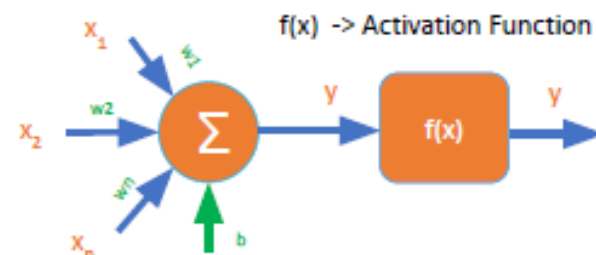
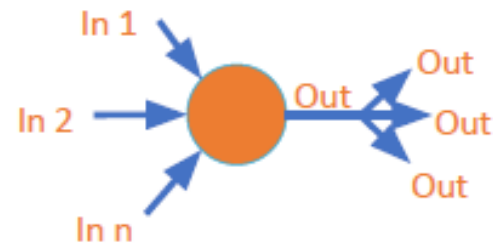
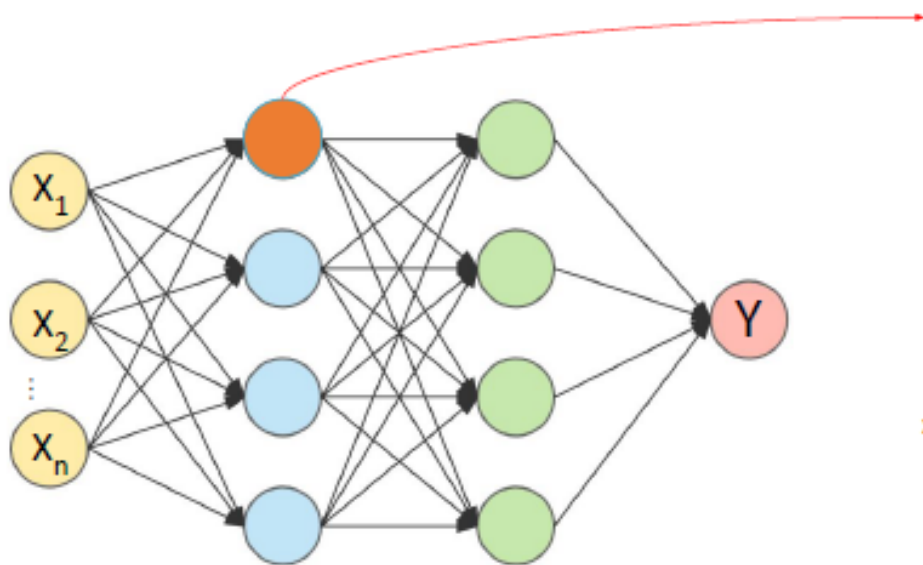
$$y = f\left(\sum_{i=1}^n x_i w_i + b\right)$$





# DNN

## ❖ Supervised machine learning with DNN



- Error Measurement (Loss)
- Optimization

Parameters to be found during training, to reach minimum error

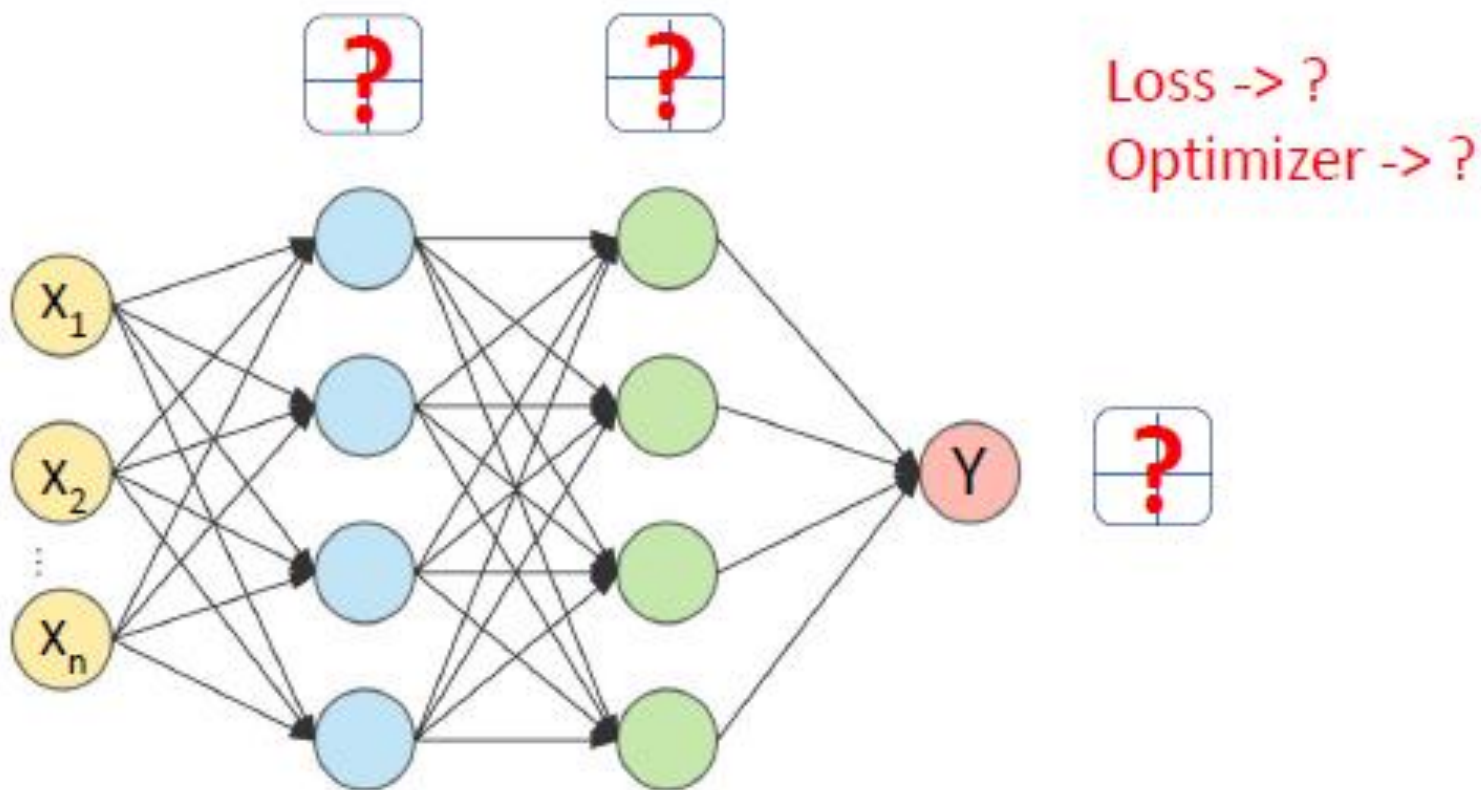
$$y = f\left(\sum_{i=1}^n x_i w_i + b\right)$$





# DNN

## ❖ Supervised machine learning with DNN

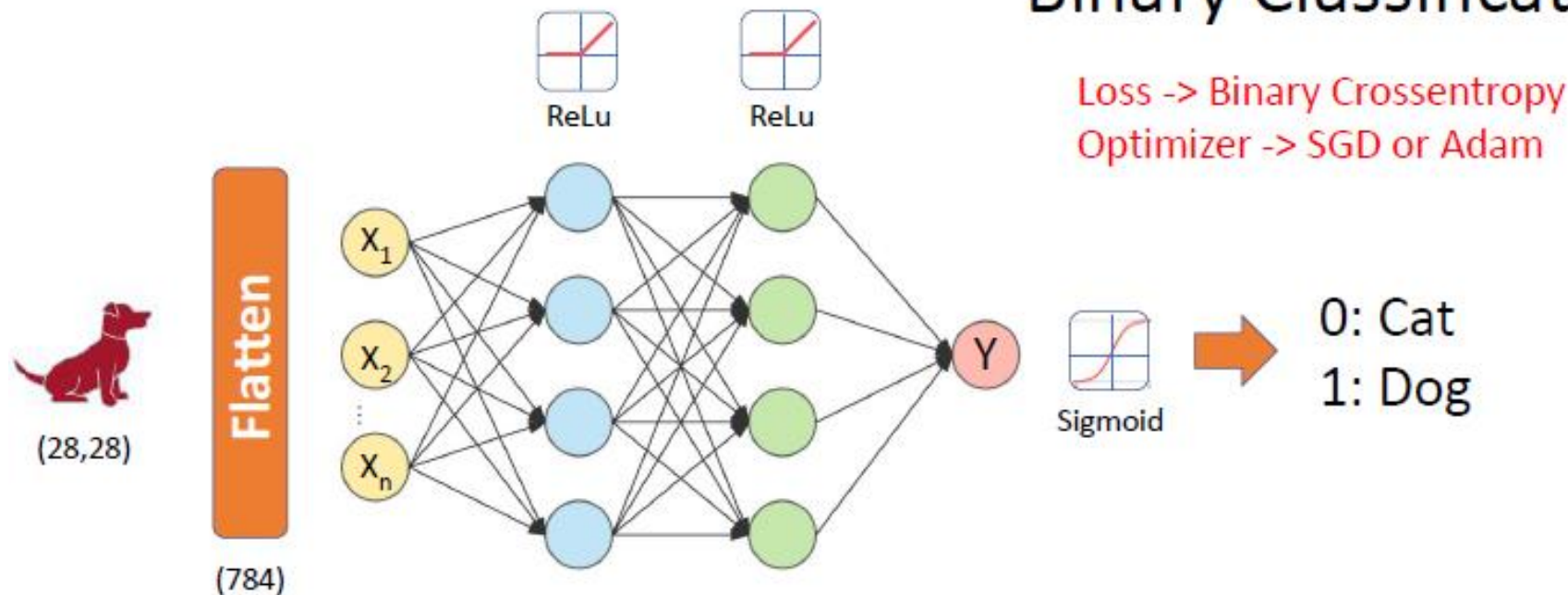


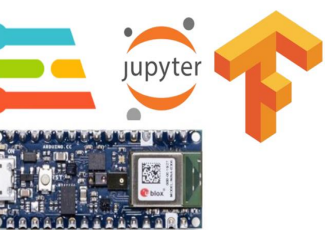


# DNN

## ❖ Supervised machine learning with DNN

### Binary Classification

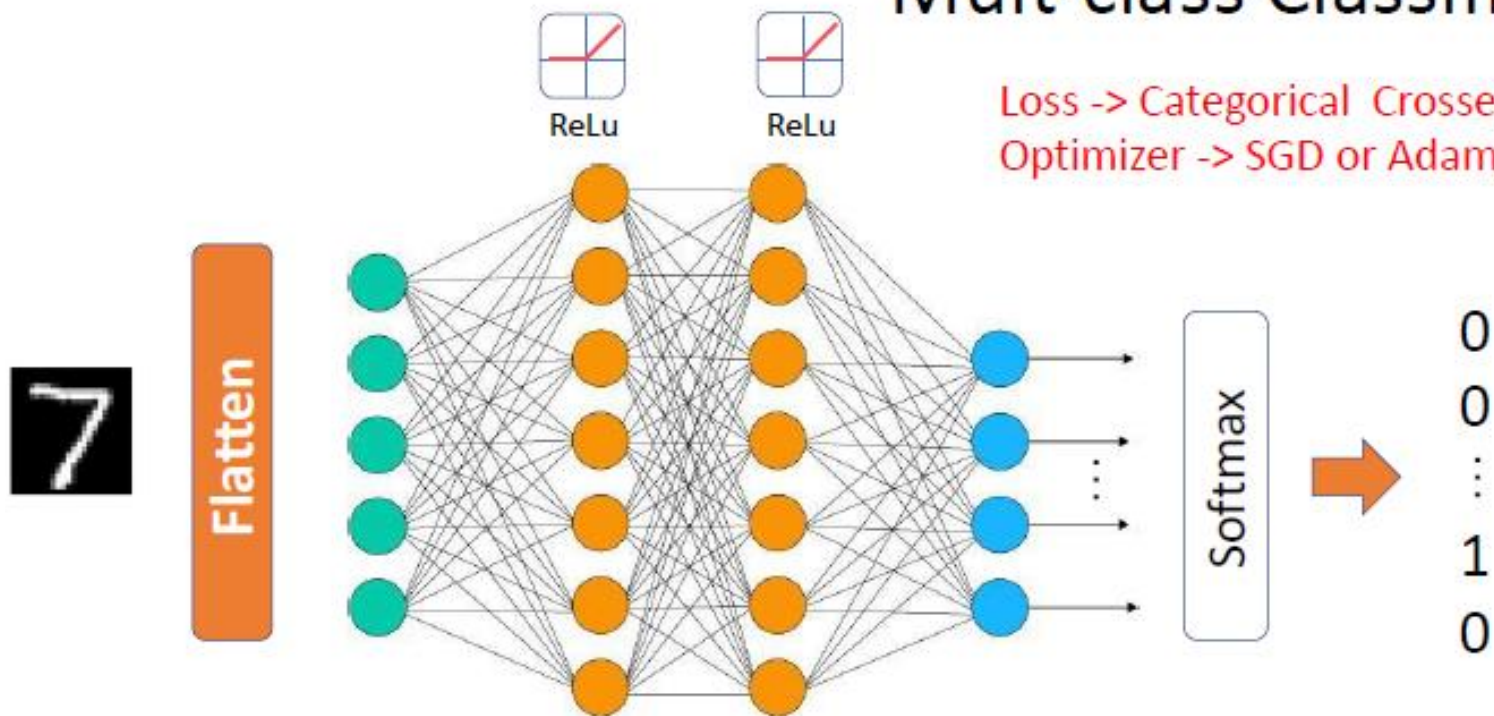




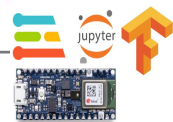
# DNN

- ❖ Supervised machine learning with DNN

## Mult-class Classification



\* or "Sparse Categorical Crossentropy" if label is 1, 2, 3, ...



## ❖ Supervised machine learning with DNN

- Dataset for training and testing



Classifying Shoes





# DNN

- ❖ Supervised machine learning with DNN
  - Dataset for training and testing

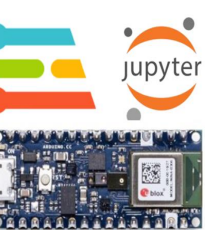
## Steps to take

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

## ❖ Supervised machine learning with DNN

- Dataset for training and testing





# DNN

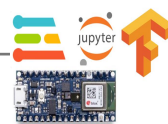
## ❖ Supervised machine learning with DNN

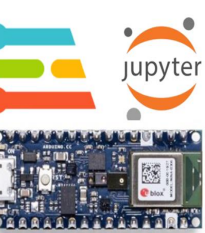
- Dataset for training and testing

## 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
```





# DNN

- ❖ Supervised machine learning with DNN
  - Dataset for training and testing

## Steps to take

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







# DNN

- ❖ Supervised machine learning with DNN
  - Dataset for training and testing

## Data

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

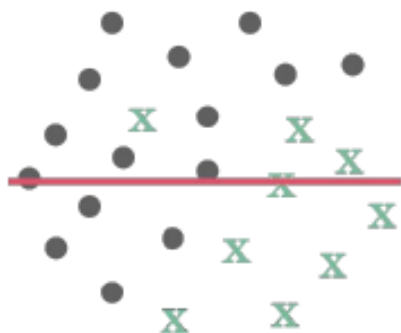
## ❖ Supervised machine learning with DNN

- Dataset for training and testing

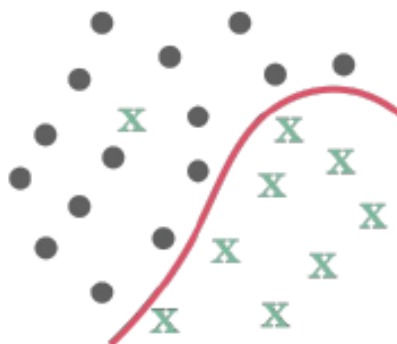
### Data

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

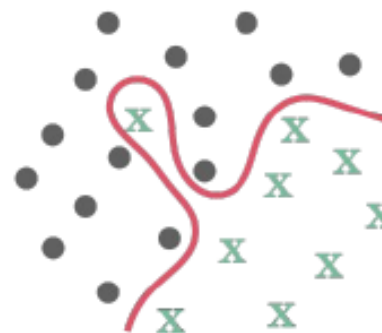
Underfitting

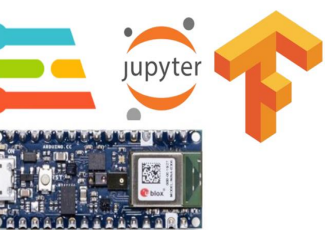


Desired



**Overfitting**





# DNN

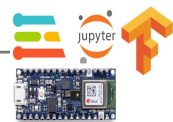
## ❖ Supervised machine learning with DNN

- Dataset for training and testing

**Data**

**Validation Data**

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

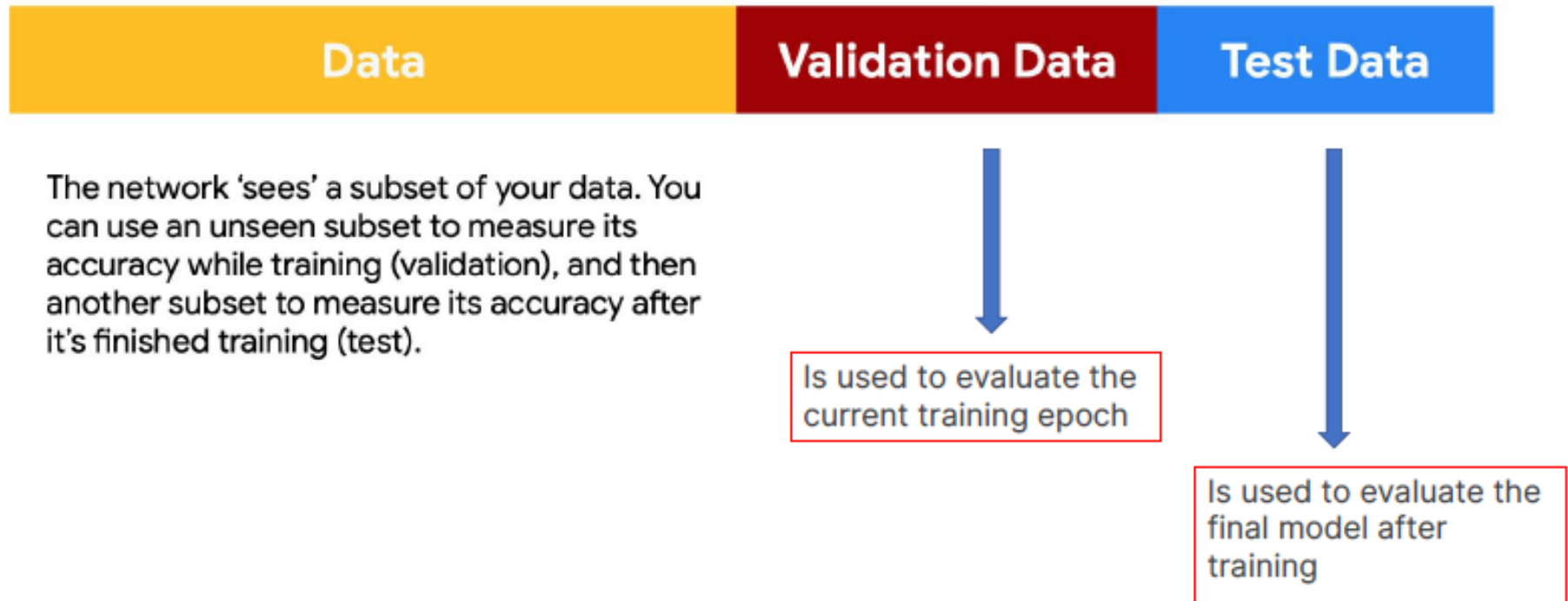




# DNN

## ❖ Supervised machine learning with DNN

- Dataset for training and testing



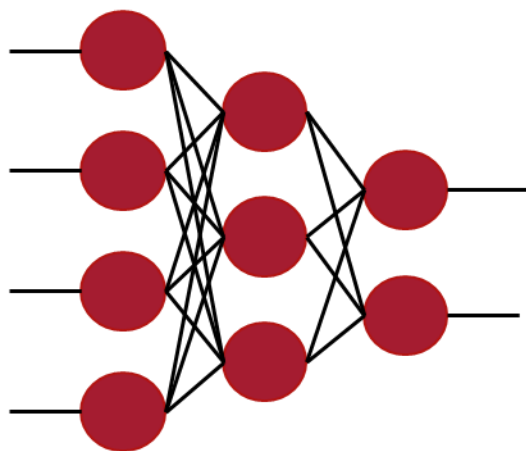


# DNN

## ❖ Supervised machine learning with DNN

- Dataset for training and testing

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



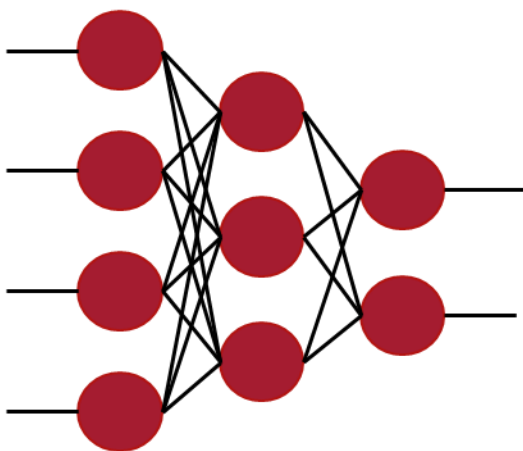


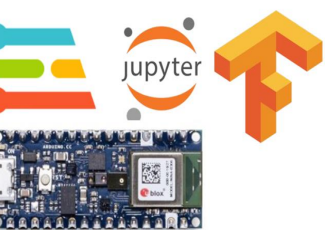
# DNN

## ❖ Supervised machine learning with DNN

- Dataset for training and testing

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



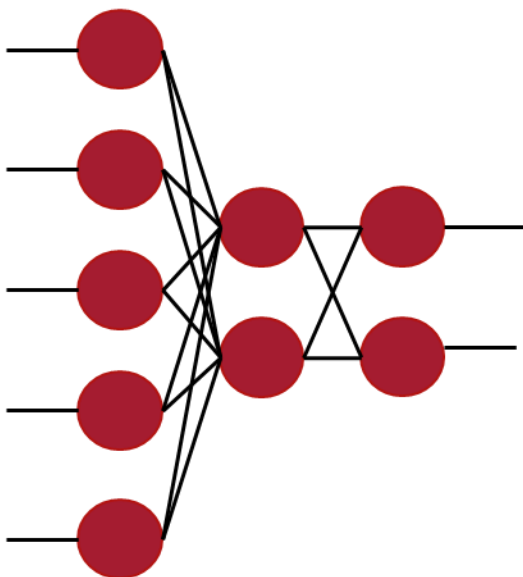


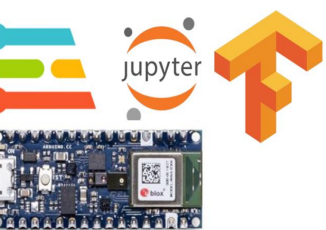
# DNN

## ❖ Supervised machine learning with DNN

- Dataset for training and testing

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





# DNN

- ❖ Supervised machine learning with DNN
  - Dataset for training and testing

## Digits Classification: validation and test dataset

Code Time!

TF\_MNIST\_Classification\_v2.ipynb







# DNN

## ❖ Supervised machine learning with DNN

- Dataset for training and testing

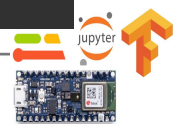
```
1 data = tf.keras.datasets.mnist
2
3 (tt_images, tt_labels), (test_images, test_labels) = data.load_data()
```

```
1 print(tt_images.shape)
2 print(tt_labels.shape)
```

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

```
1 print(test_images.shape)
2 print(test_labels.shape)
```

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





# DNN

## ❖ Supervised machine learning with DNN

### □ Dataset for training and testing

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

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

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

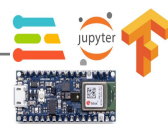
```
(50000, 28, 28)
(50000,)
```

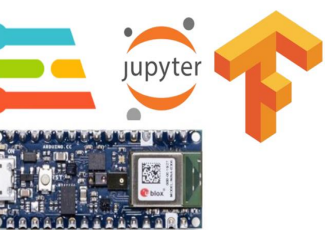
```
1 print(val_images.shape)
2 print(val_labels.shape)
```

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

Split tt data in:

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





# DNN

## ❖ Supervised machine learning with DNN

- Dataset for training and testing

```
1 history = model.fit(  
2     train_images,  
3     train_labels,  
4     epochs=20,  
5     validation_data=(val_images, val_labels)  
6 )
```

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.

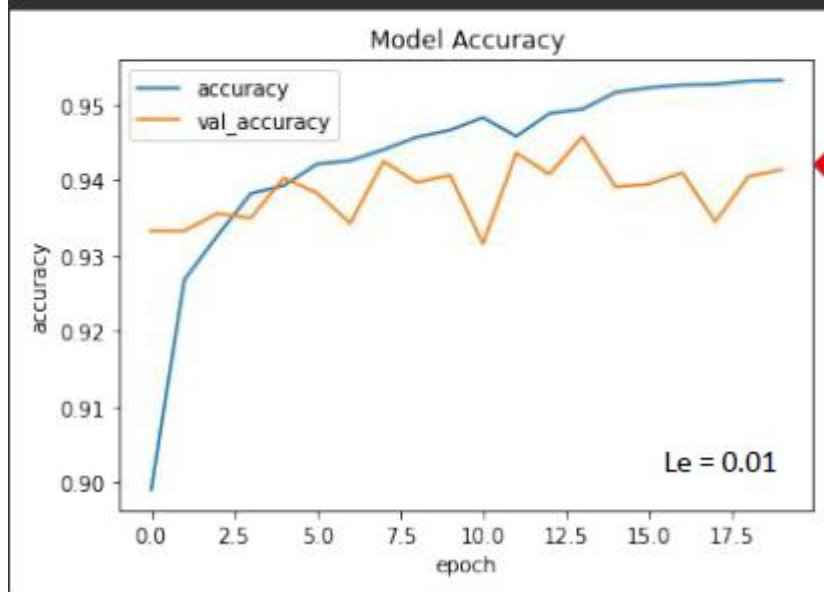


# DNN

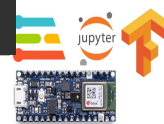
## ❖ Supervised machine learning with DNN

### □ Dataset for training and testing

```
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 “instable”, could be that Learning Rate is high (try to reduce it).



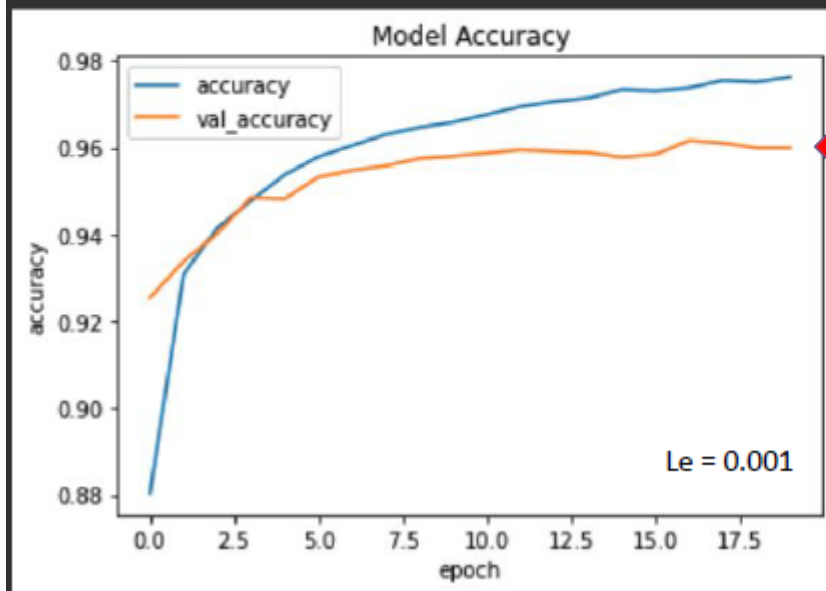


# DNN

## ❖ Supervised machine learning with DNN

### □ Dataset for training and testing

```
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.

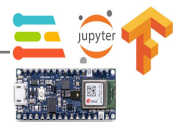


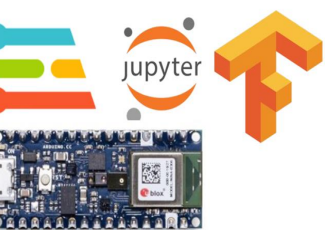
# DNN

- ❖ Supervised machine learning with DNN
  - Dataset for training and testing

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

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

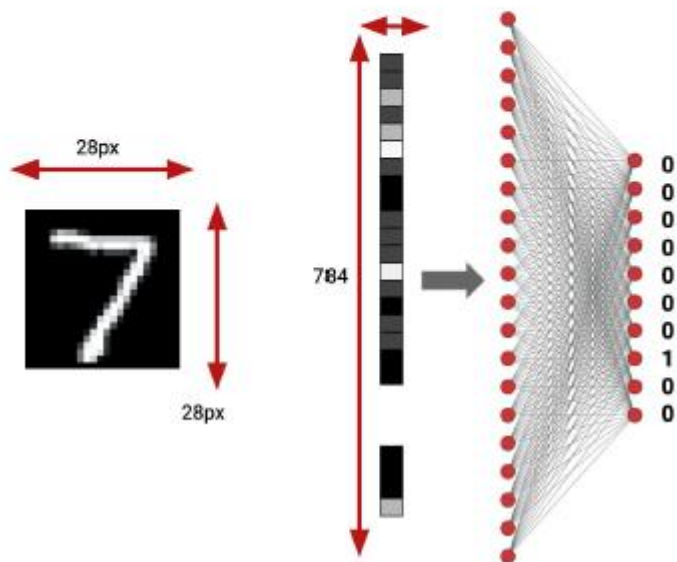
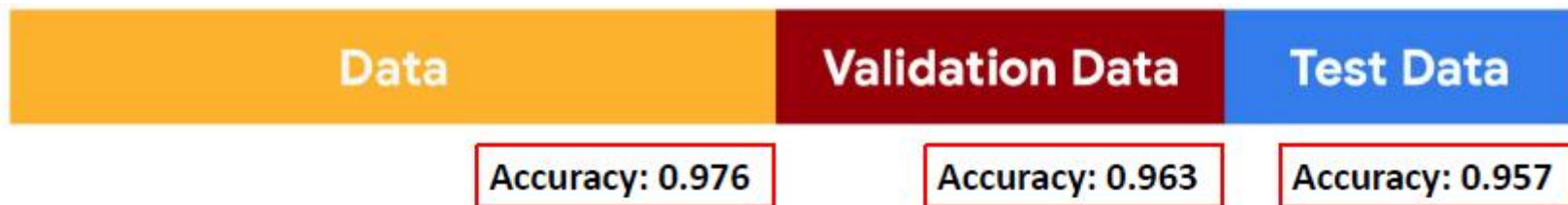




# DNN

## ❖ Supervised machine learning with DNN

- Dataset for training and testing





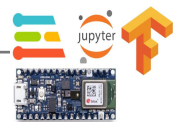


# DNN

## ❖ Supervised machine learning with DNN

### □ Dataset for training and testing

- 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







# MODEL PERFORMANCE METRICS

## ❖ Classification model performance metrics



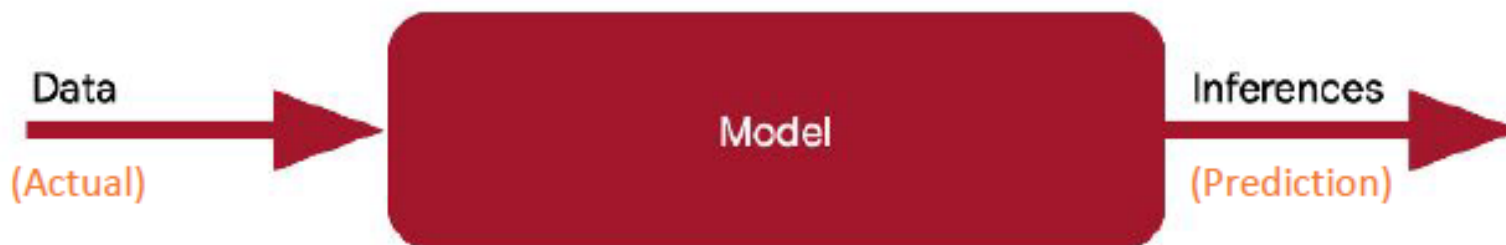
Class = [1]

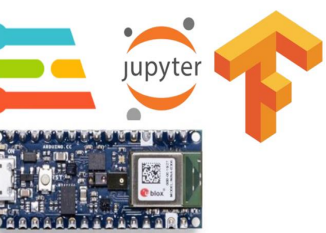
actual = [1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0]



Class = [0]

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





# MODEL PERFORMANCE METRICS

## ❖ Classification model performance metrics



Class = [1]

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

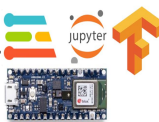


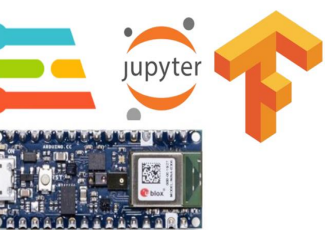
Class = [0]

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

## Model Performance (Confusion Matrix)

		predicted condition	
		Cat [1]	Dog [0]
true condition	Cat [1]	6	2
	Dog [0]	1	3





# MODEL PERFORMANCE METRICS

## ❖ Classification model performance metrics



Class = [1]

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

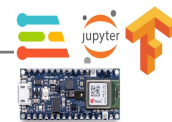


Class = [0]

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

## Model Performance (Confusion Matrix)

		predicted condition	
		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



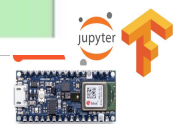


# MODEL PERFORMANCE METRICS

- ❖ Classification model performance metrics

## Model Performance (Confusion Matrix)

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



# MODEL PERFORMANCE METRICS

## ❖ Classification model performance metrics

**Type I error (false positive)**



**Type II error (false negative)**





# MODEL PERFORMANCE METRICS

## ❖ Classification model performance metrics

### 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



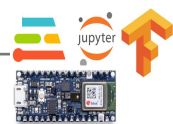
Low Precision, High Accuracy



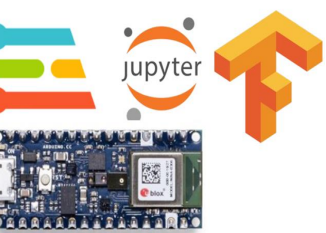
High Precision, Low Accuracy



Low Precision, Low Accuracy







# MODEL PERFORMANCE METRICS

## ❖ Classification model performance metrics

### Accuracy , Precision and Recall

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

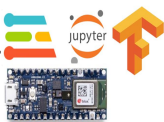
$$\text{Precision} = \frac{TP}{(TP + FP)} = \frac{6}{(6 + 1)} = \frac{6}{7} = 0.86$$

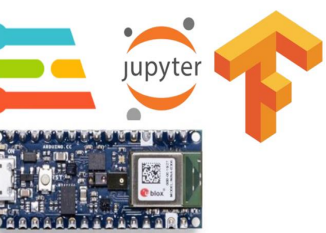
$\frac{\text{Total Positive}}{\text{Total Predict Positive}}$

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

(or Sensitivity)

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





# MODEL PERFORMANCE METRICS

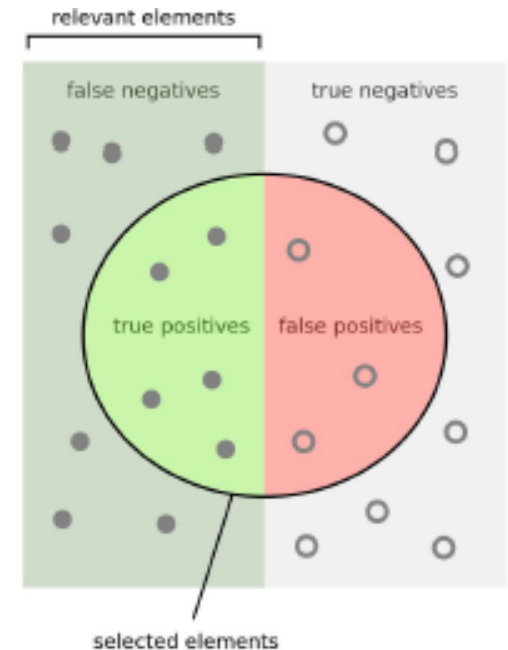
## ❖ Classification model performance metrics

### 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

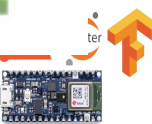


How many selected items are relevant?

Precision =  $\frac{\text{green}}{\text{green} + \text{pink}}$

How many relevant items are selected?

Recall =  $\frac{\text{green}}{\text{green}}$





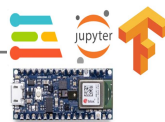


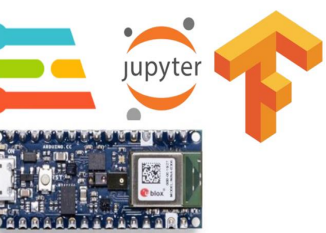
# MODEL PERFORMANCE METRICS

- ❖ Classification model performance metrics

Classification Report  
Code Time!

Classification\_Report.ipynb





# MODEL PERFORMANCE METRICS

## ❖ Classification model performance metrics

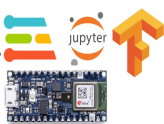
```
1 from sklearn.metrics import classification_report
```

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

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

```
1 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



## Model Performance (Confusion Matrix)

		predicted condition	
		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

## Confusion Matrix

	12 pictures, 8 of cats and 4 of dogs	Predicted condition	
		Cat[1]	Dog[0]
True condition	Cat[1]	<b>TP = 6</b>	<b>FN = 2</b>
	Dog[0]	<b>FP = 1</b>	<b>TN = 3</b>