IESTI01 - TinyML

The Building Blocks of Deep Learning – Part C

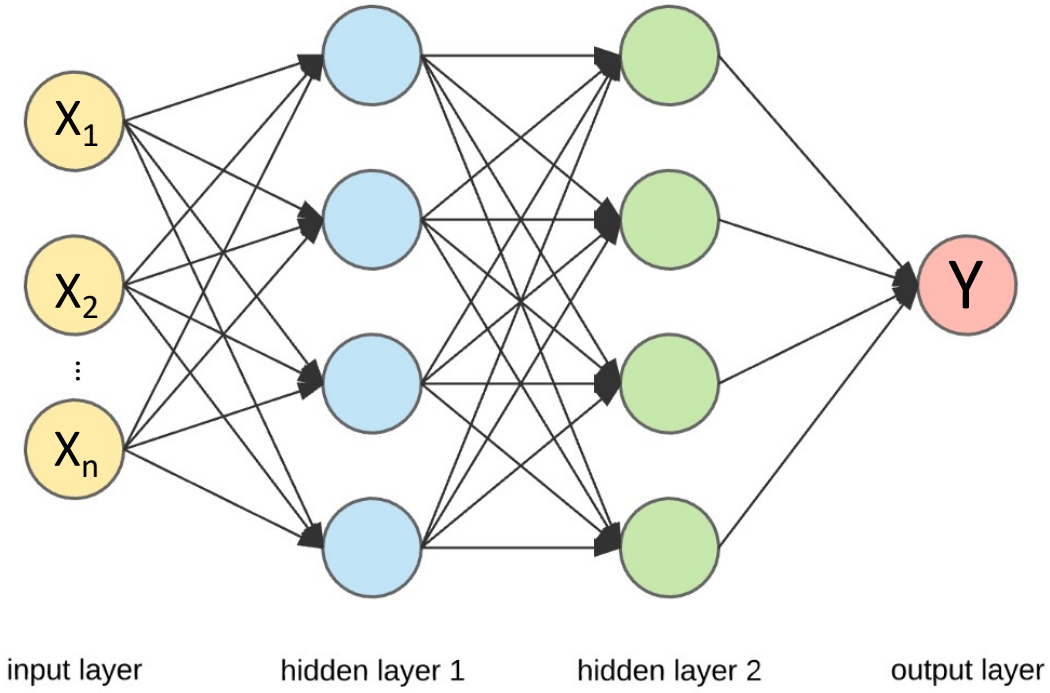
Prof. Marcelo Rovai

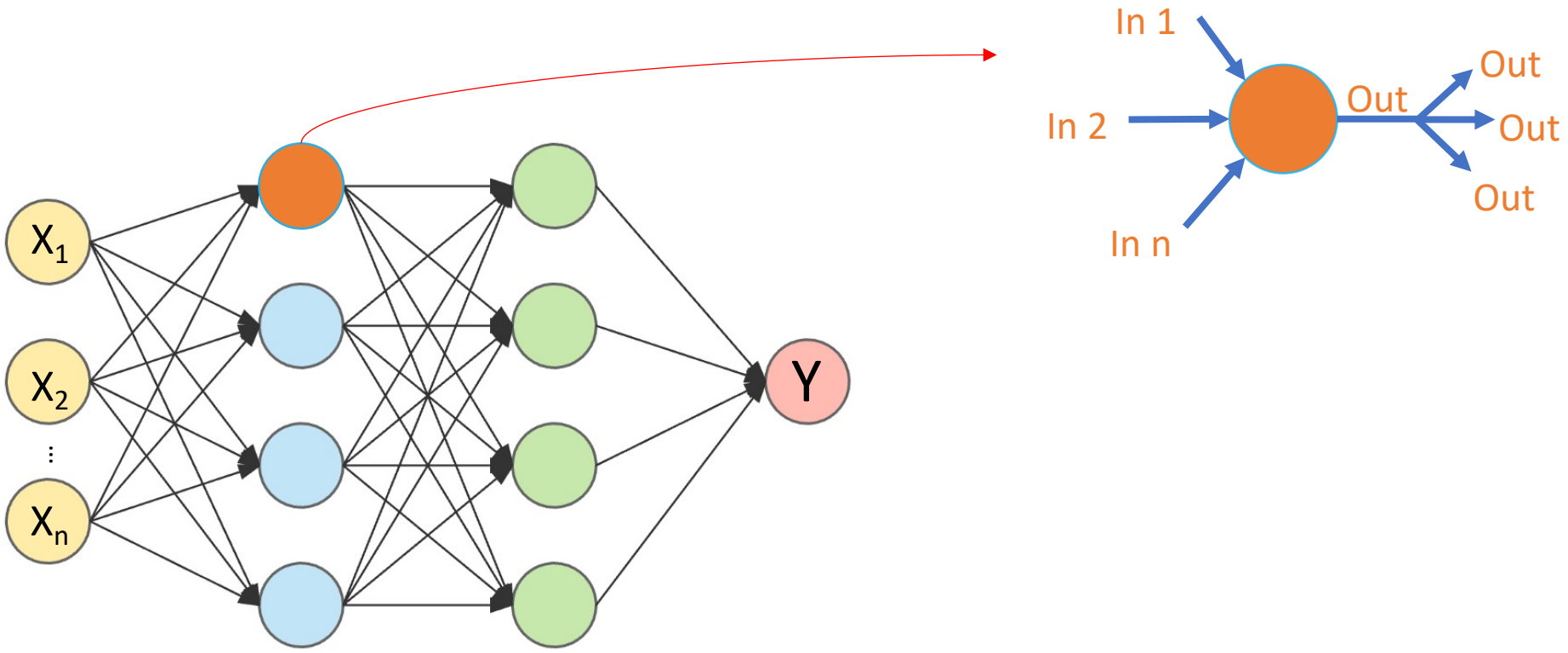
May 26th, 2021

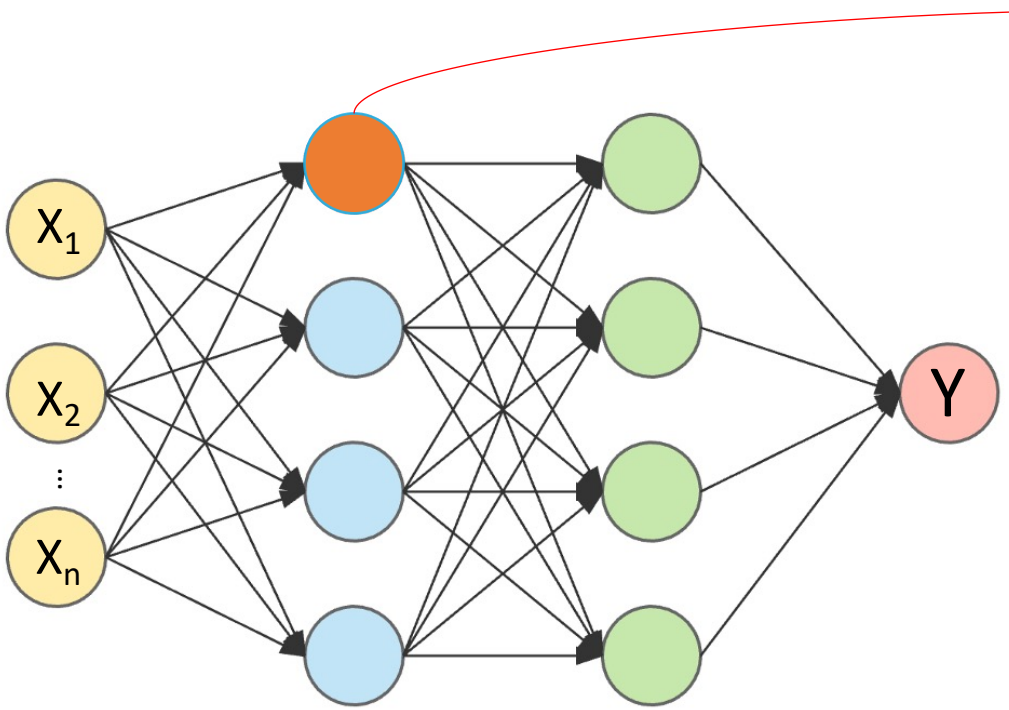
DNN Dense Neural Network

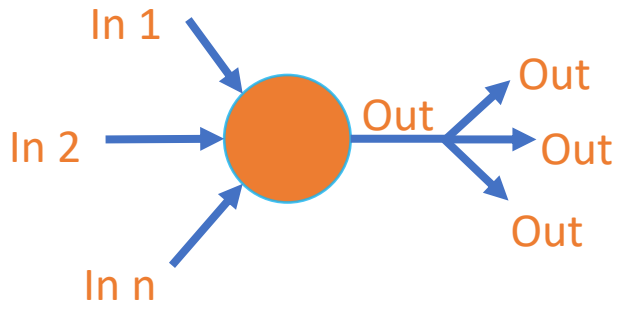
Recap

Supervised Machine Learning with DNN

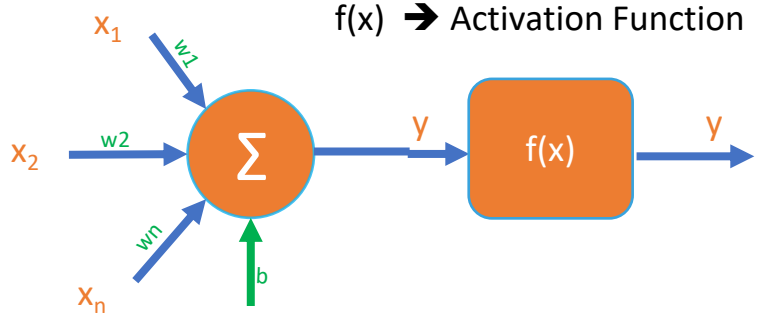



X1
X2
Xn
Y
In 1
In 2
In n
Out
Out
Out
Out



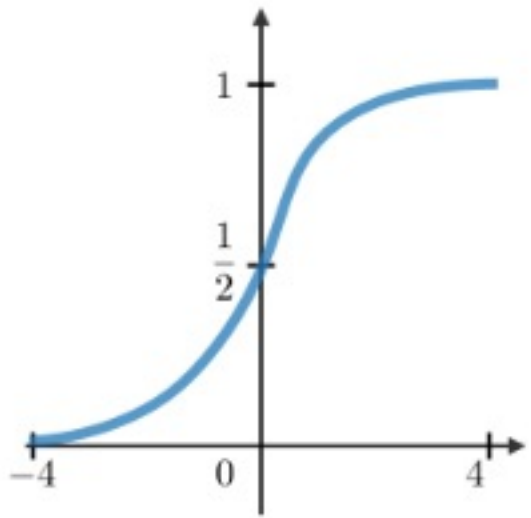
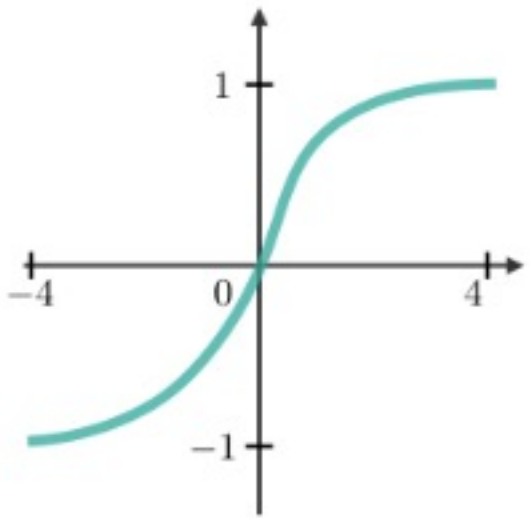
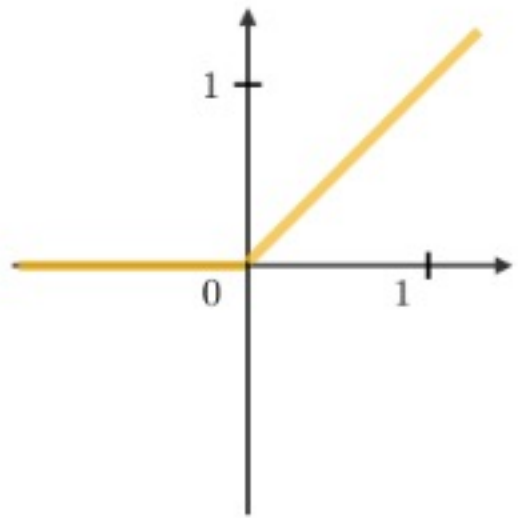


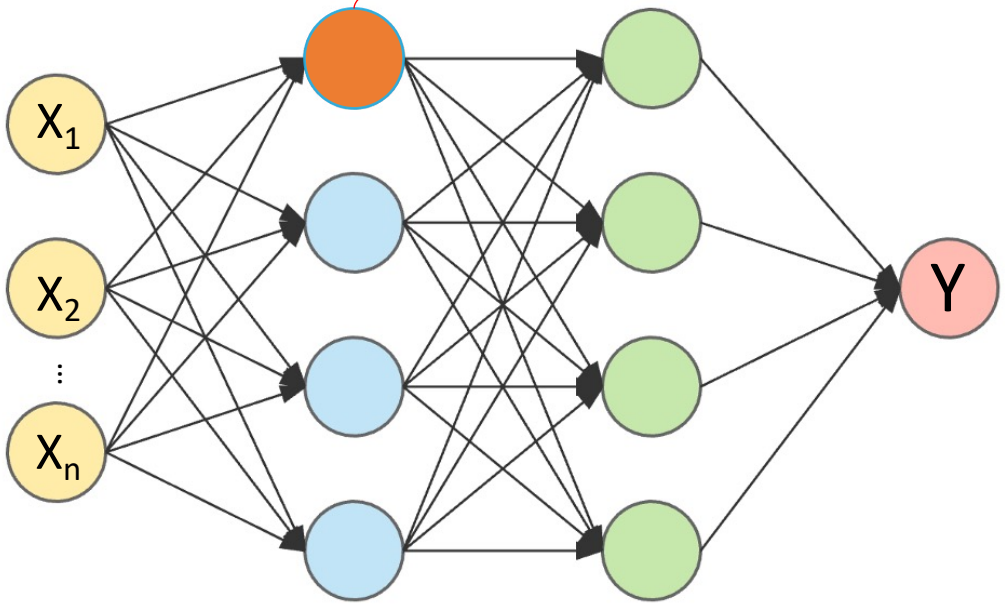
f(x) ➔ Activation Function

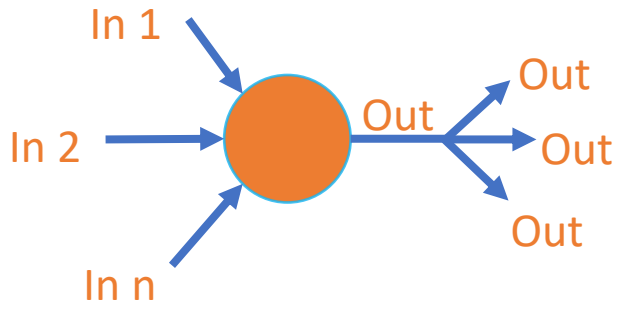


$$y = f(\sum_{i=1}^{n} x_i w_i + b)$$

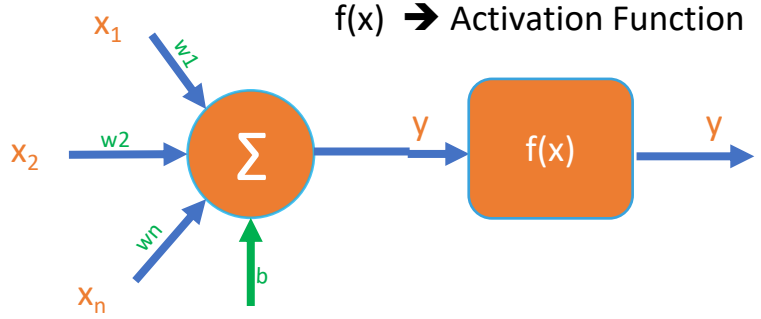
Activation Functions

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



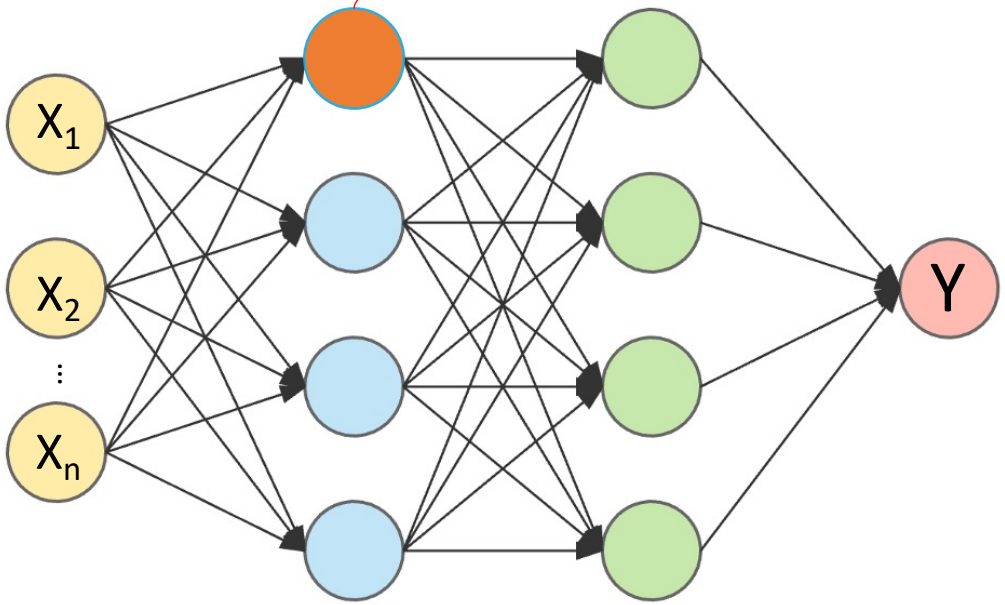


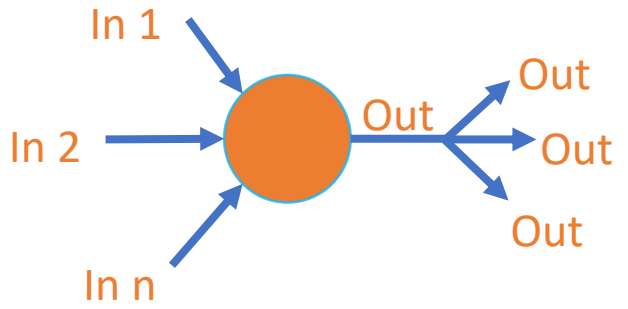
f(x) ➔ Activation Function

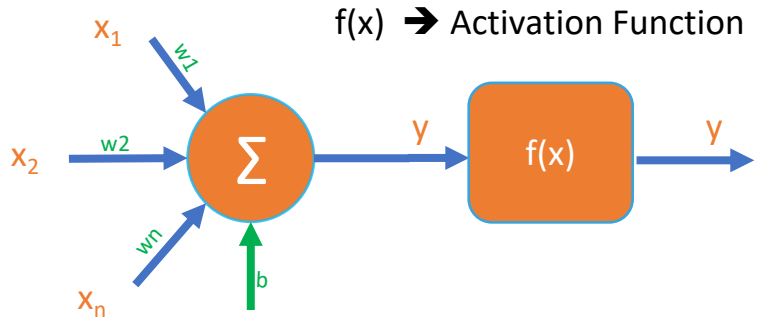


Parameters to be found during training
To reach minimum error

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



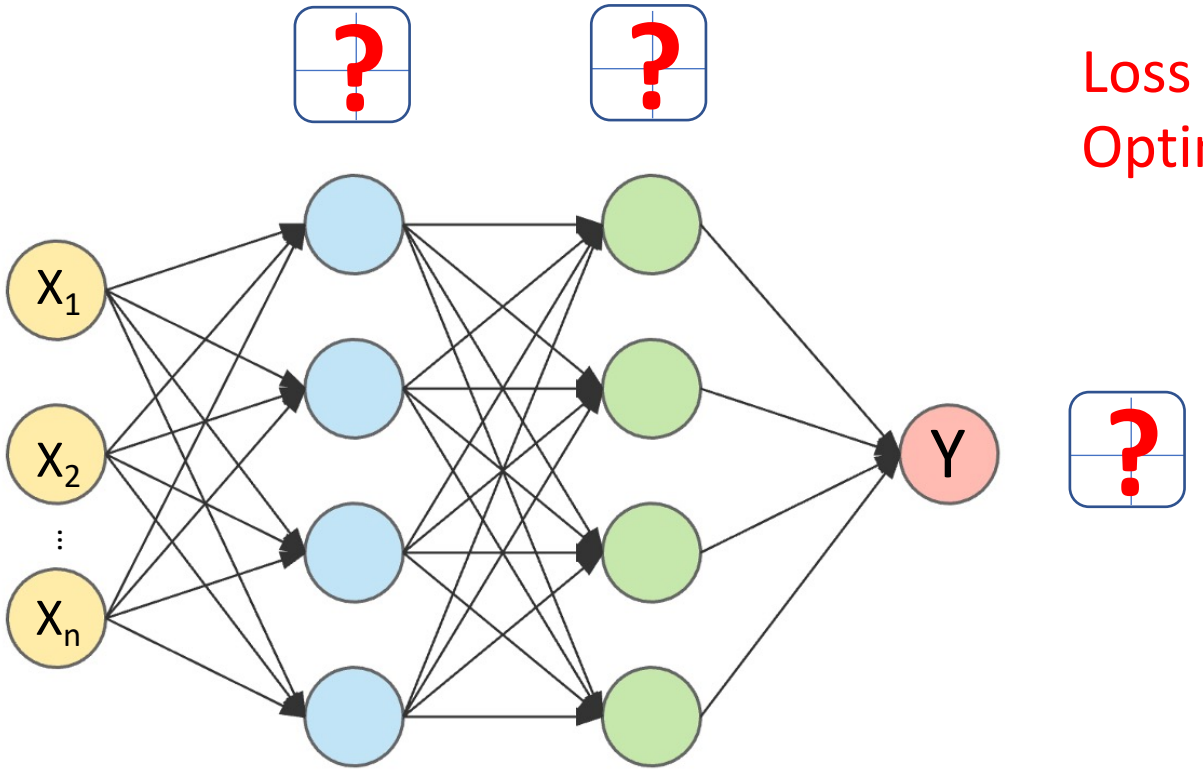




- Error Measurement (Loss)
- Optimization

Parameters to be found during training
To reach minimum error

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

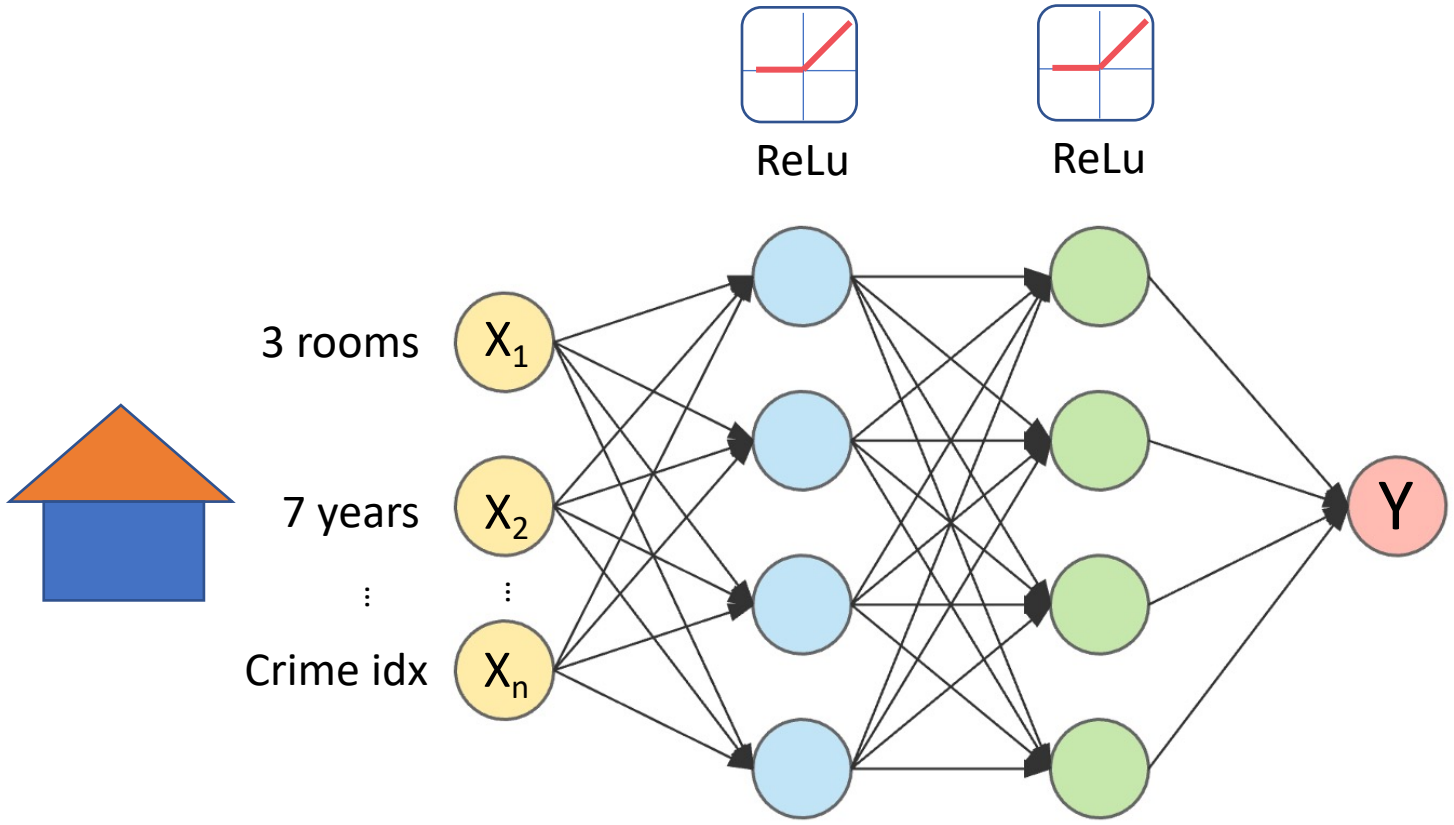


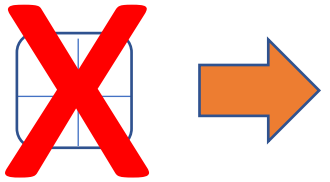
Loss ➔ ?

Optimizer ➔ ?

Regression

Loss ➔ MSE or MAE
Optimizer ➔ SGD or Adam





$34.8

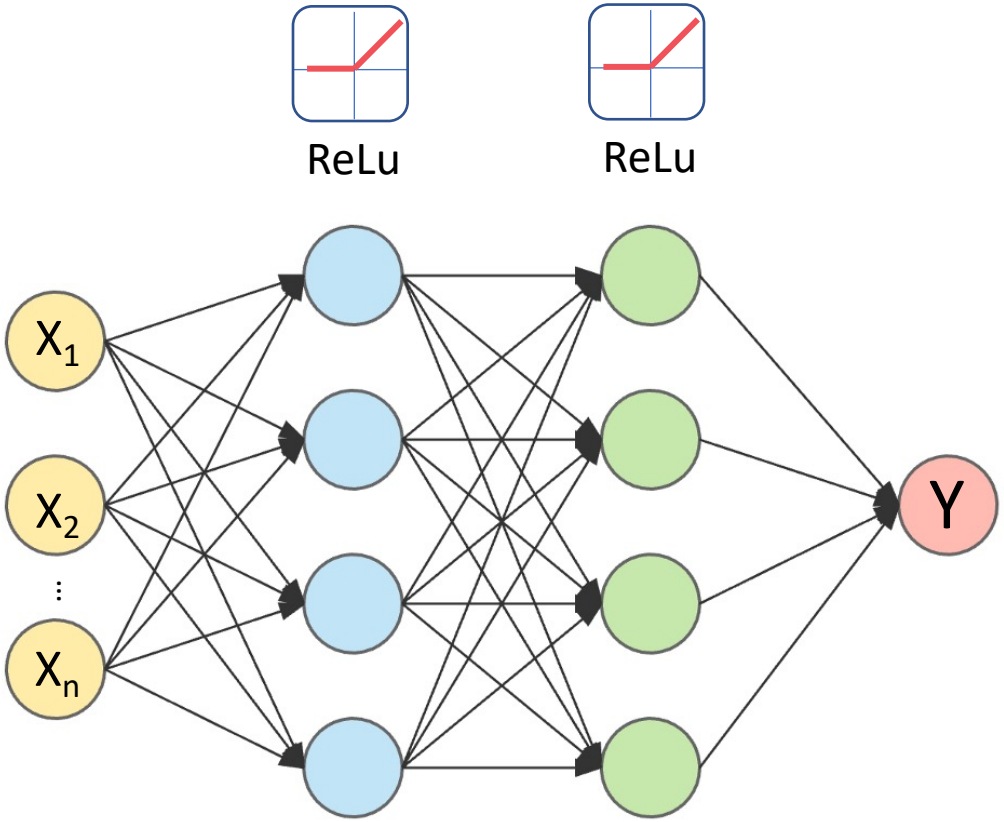
Binary Classification

Loss ➔ Binary Crossentropy
Optimizer ➔ SGD or Adam

(28,28)

Flatten

(784)

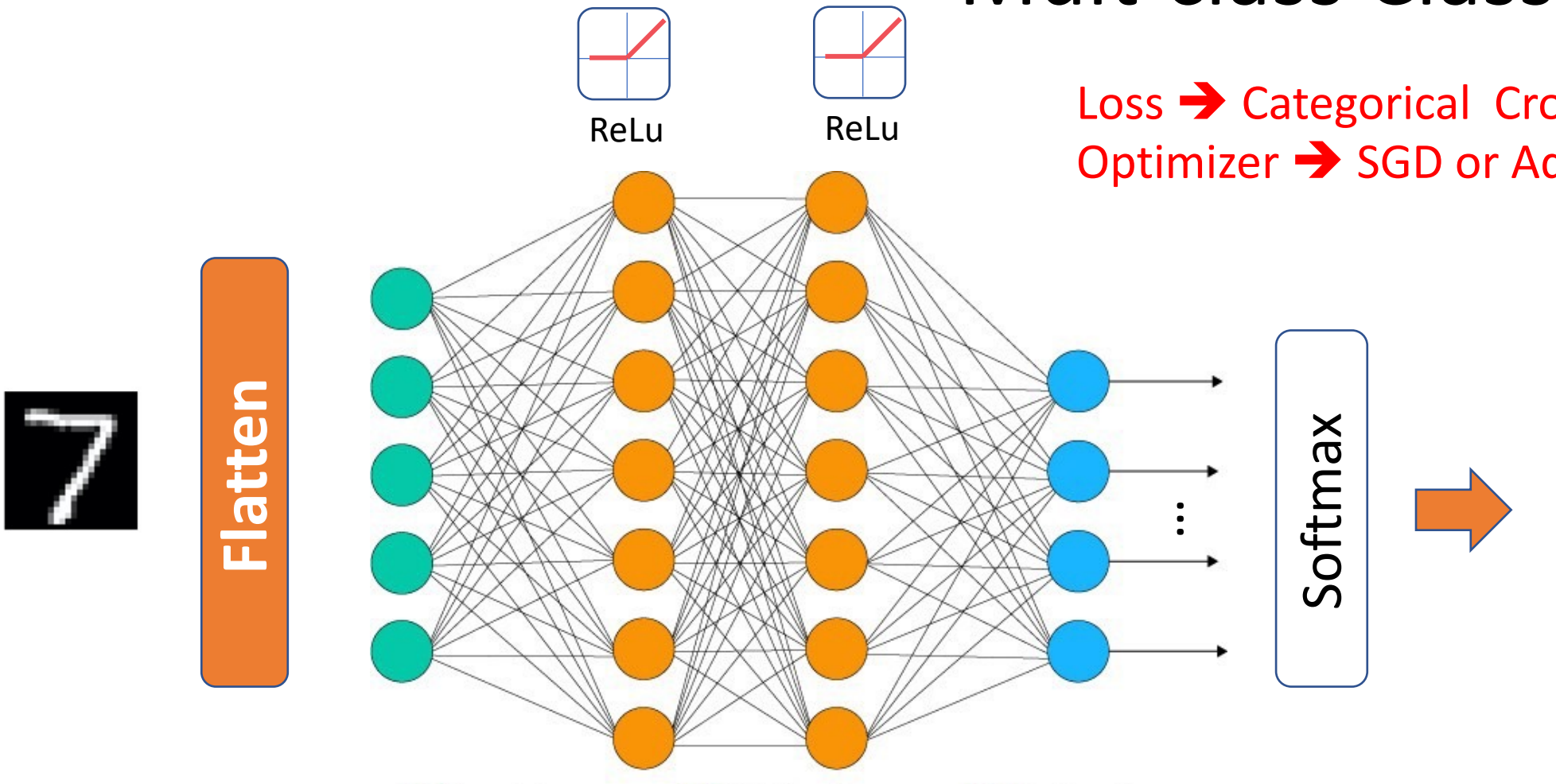


Sigmoid

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

Classifing Shoes

Steps to take

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


FINLAND

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?

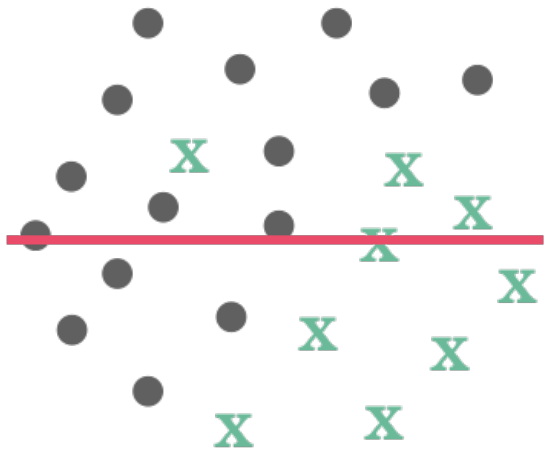
Data

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

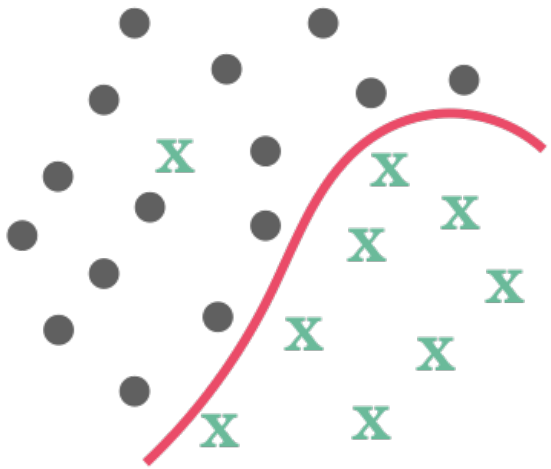
Data

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

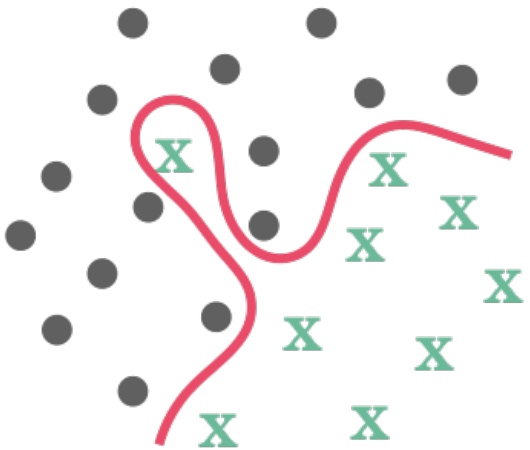
Underfitting



Desired



Overfitting



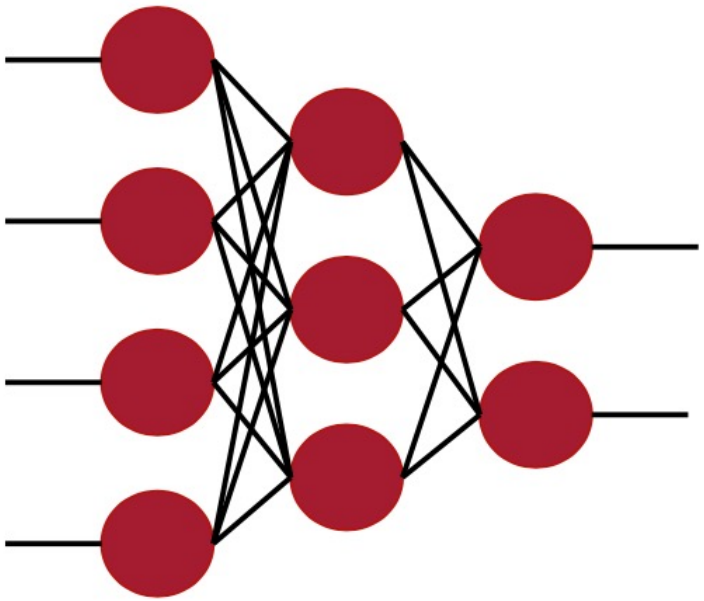
The network ‘sees’ a subset of your data. You can use the rest to measure its performance against previously unseen data.

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).

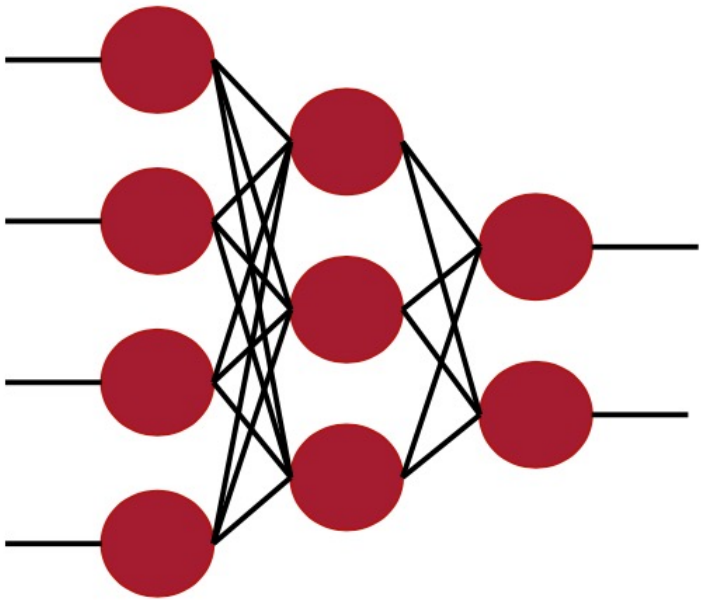
Is used to evaluate the current training epoch

Is used to evaluate the final model after training

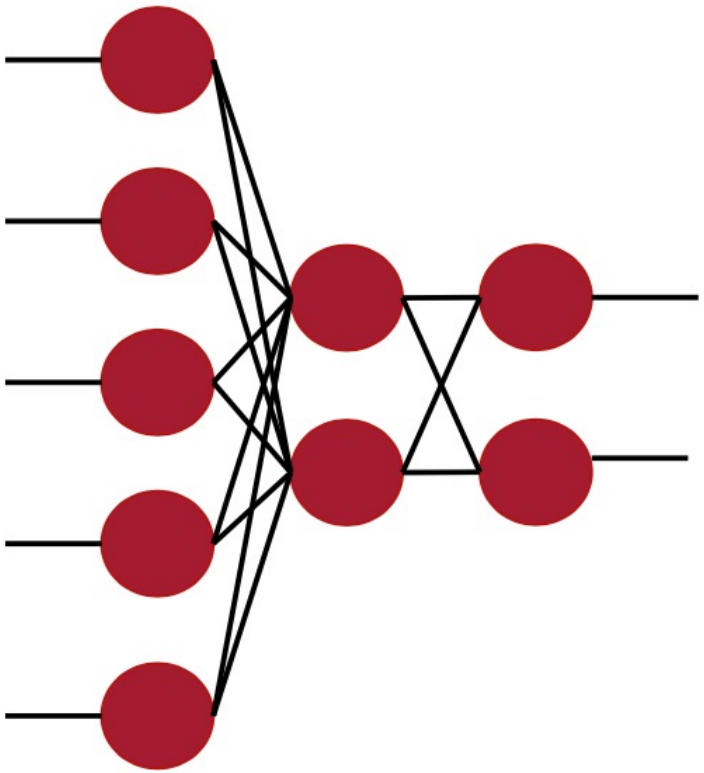
Data	Validation Data	Test Data
Accuracy: 0.999	Accuracy: 0.920	Accuracy: 0.800



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



```
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)])
```

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

(tt_images, tt_labels), (val_images, val_labels) = data.load_data()
```

```
test_images = tt_images[:10000]
test_labels = tt_labels[:10000]
```

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

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

```
training_images = tt_images[10000:]
training_labels = tt_labels[10000:]
```

```
print(training_images.shape)
print(training_labels.shape)
```

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

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

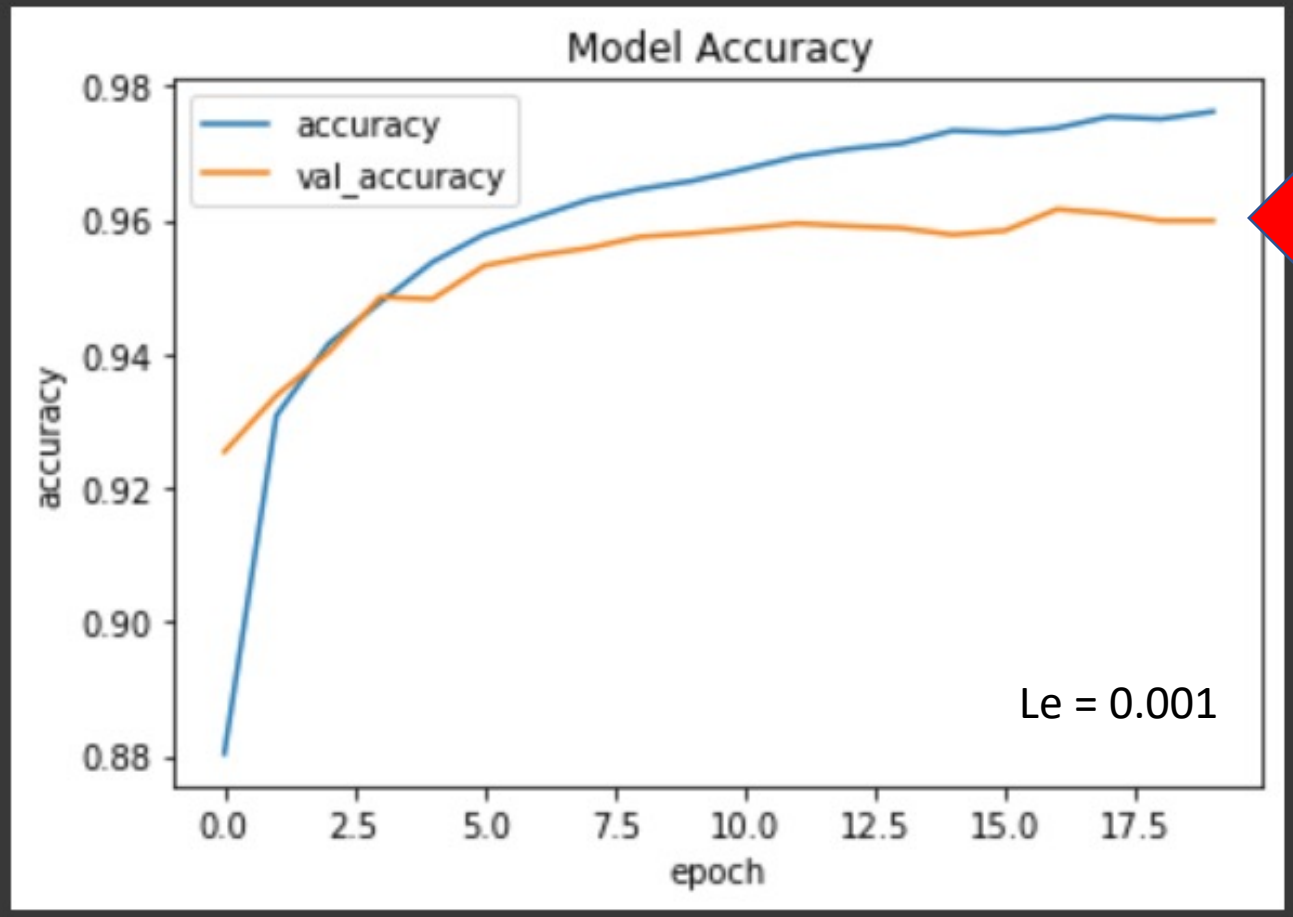
```
Epoch 16/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0931 - accuracy: 0.9729
Epoch 17/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0907 - accuracy: 0.9731
Epoch 18/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0892 - accuracy: 0.9735
Epoch 19/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0868 - accuracy: 0.9740
Epoch 20/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0844 - accuracy: 0.9750
```

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

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

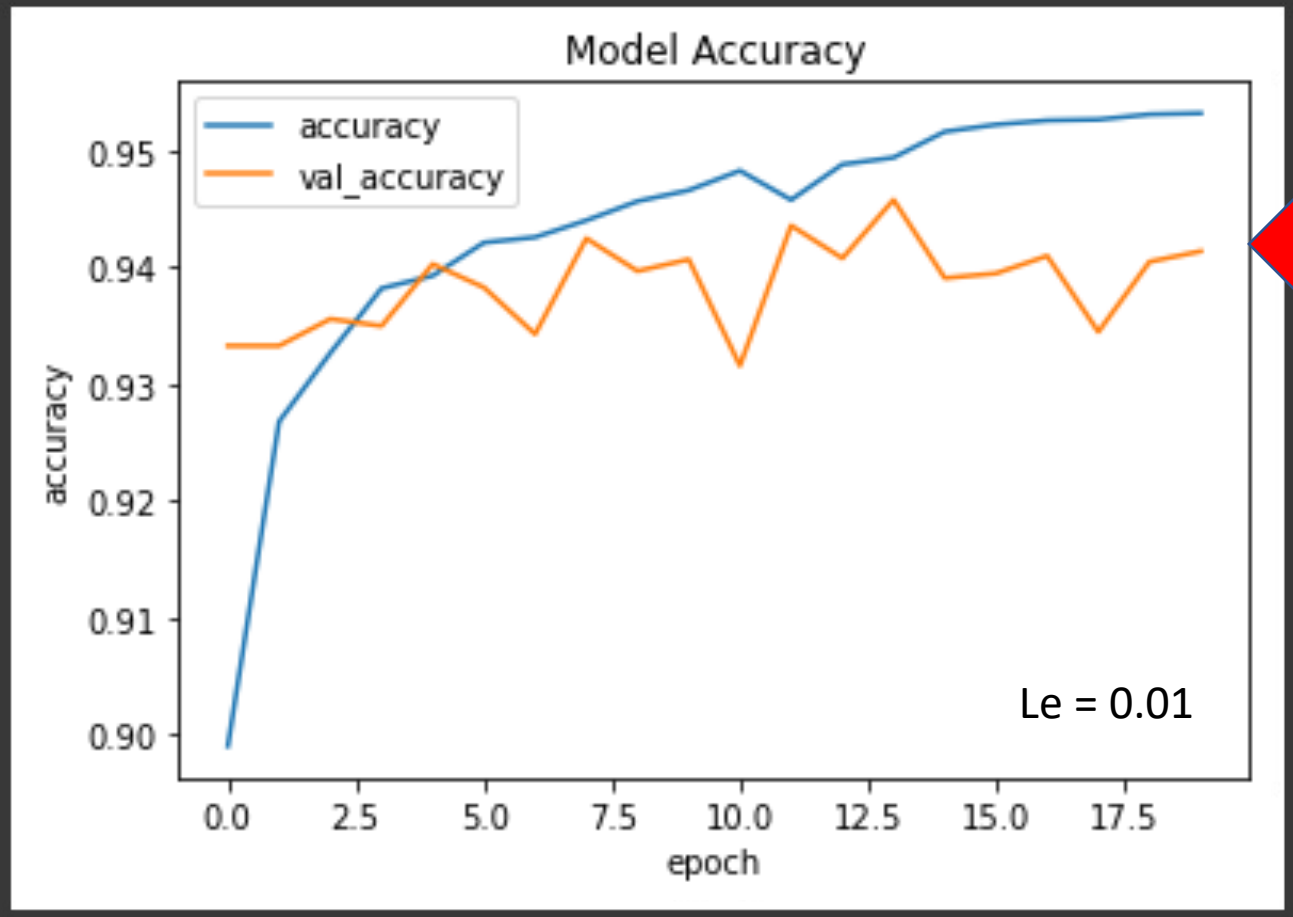
```
Epoch 14/20
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0960 - accuracy: 0.9713 - val_loss: 0.1410 - val_accuracy: 0.9588
Epoch 15/20
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0933 - accuracy: 0.9722 - val_loss: 0.1353 - val_accuracy: 0.9626
Epoch 16/20
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0910 - accuracy: 0.9729 - val_loss: 0.1356 - val_accuracy: 0.9622
Epoch 17/20
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0880 - accuracy: 0.9734 - val_loss: 0.1339 - val_accuracy: 0.9625
Epoch 18/20
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0855 - accuracy: 0.9740 - val_loss: 0.1349 - val_accuracy: 0.9619
Epoch 19/20
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0832 - accuracy: 0.9754 - val_loss: 0.1341 - val_accuracy: 0.9624
Epoch 20/20
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0808 - accuracy: 0.9759 - val_loss: 0.1340 - val_accuracy: 0.9626
```

```
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, even if train accuracy goes up, means that probably the model is overfitting. In this case the training (epochs) should terminate.

```
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()
```

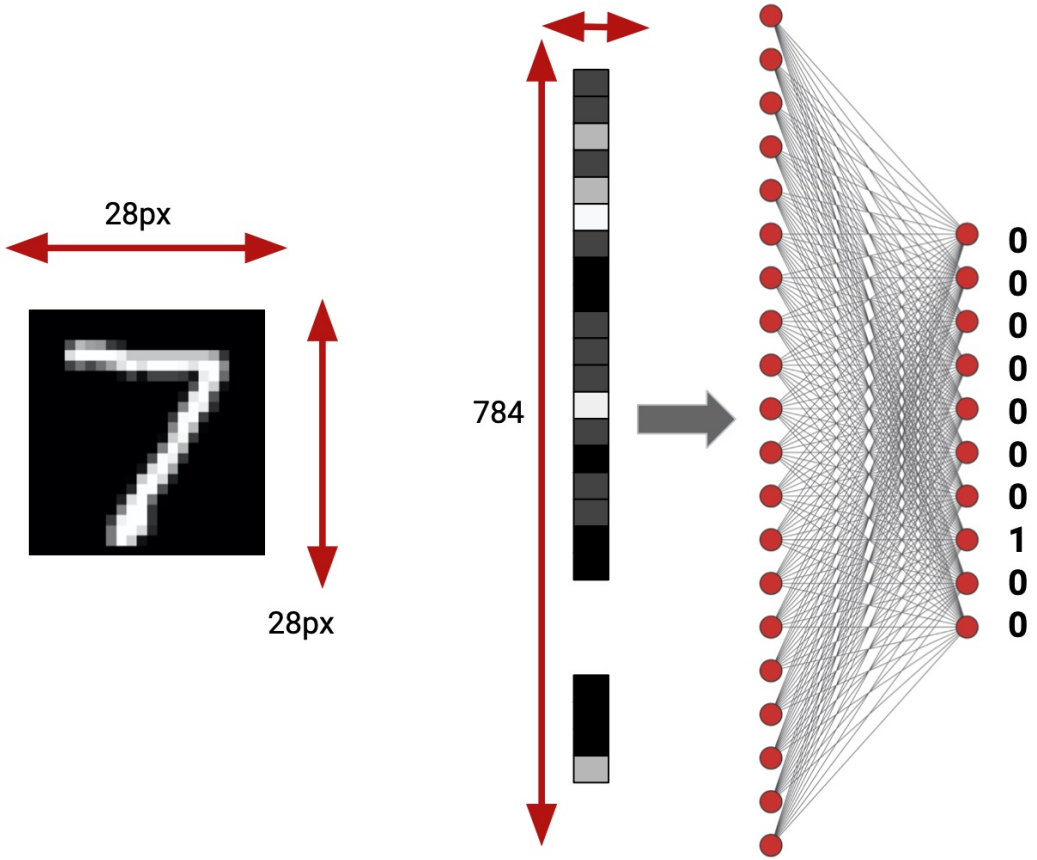


If validation accuracy seems “instable”, could be that Learning Rate is high (try to reduce it).

```
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

Digits Classification: validation and test dataset

Code Time!

TF_MNIST_Classification_v2.ipynb

Going Further

Classification Model Performance Metrics

Class = [1]

Class = [0]

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]

Model Performance (Confusion Matrix)

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

Model Performance (Confusion Matrix)

		predicted condition	
	12 pictures, 8 of cats and 4 of dogs	**Cat**	**Dog**
true condition	**Cat**	**True Positive (TP)** 6	**False Negative (FN)** (type II error) 2
	Dog	**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

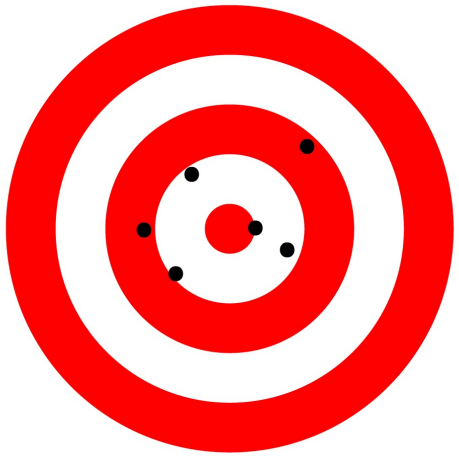
Type I error (false positive)

Type II error (false negative)

Precision vs. Accuracy

In a set of measurements, **accuracy** is closeness of the measurements to a specific value, while **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)

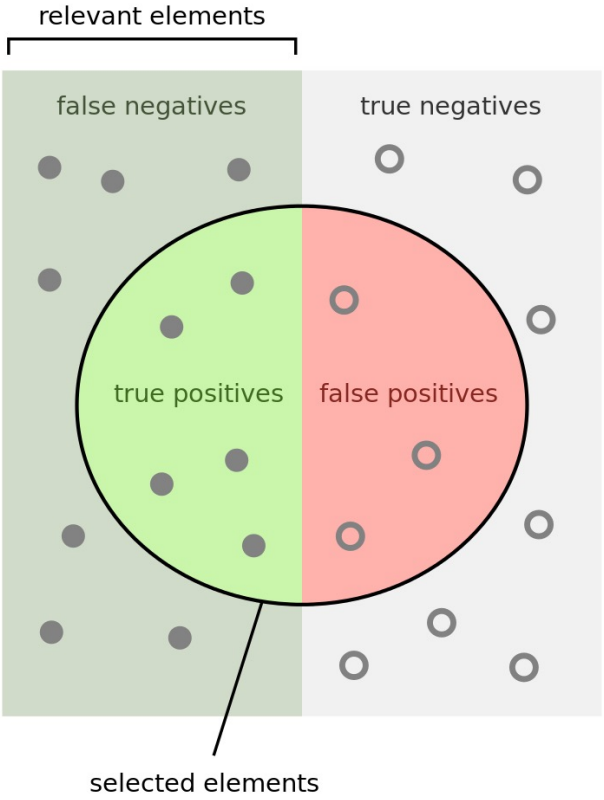
$$\frac{\text{Total Positive}}{\text{Total Actual Positive}}$$

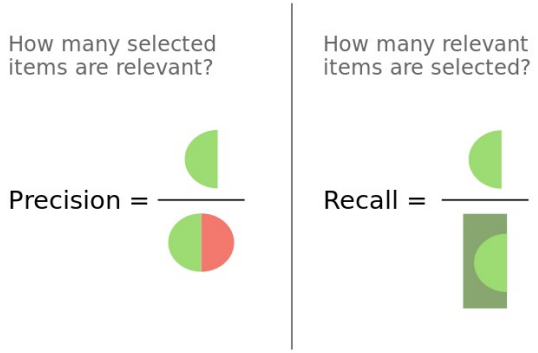
F1-Score

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

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

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





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

```
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
```

Classification Report

Code Time!

Classification_Report.ipynb

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)
- Text Book: "TinyML" by Pete Warden, Daniel Situnayake

I want to thank Laurence Moroney from Google, Harvard professor Vijay Janapa Reddi, Ph.D. student Brian Plancher and their staff 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

And stay safe!