EECE-4710 "IoT and TinyML"

Machine Learning Metrics

Cris Ababei

MARQUETTE UNIVERSITY

BE THE DIFFERENCE.

1

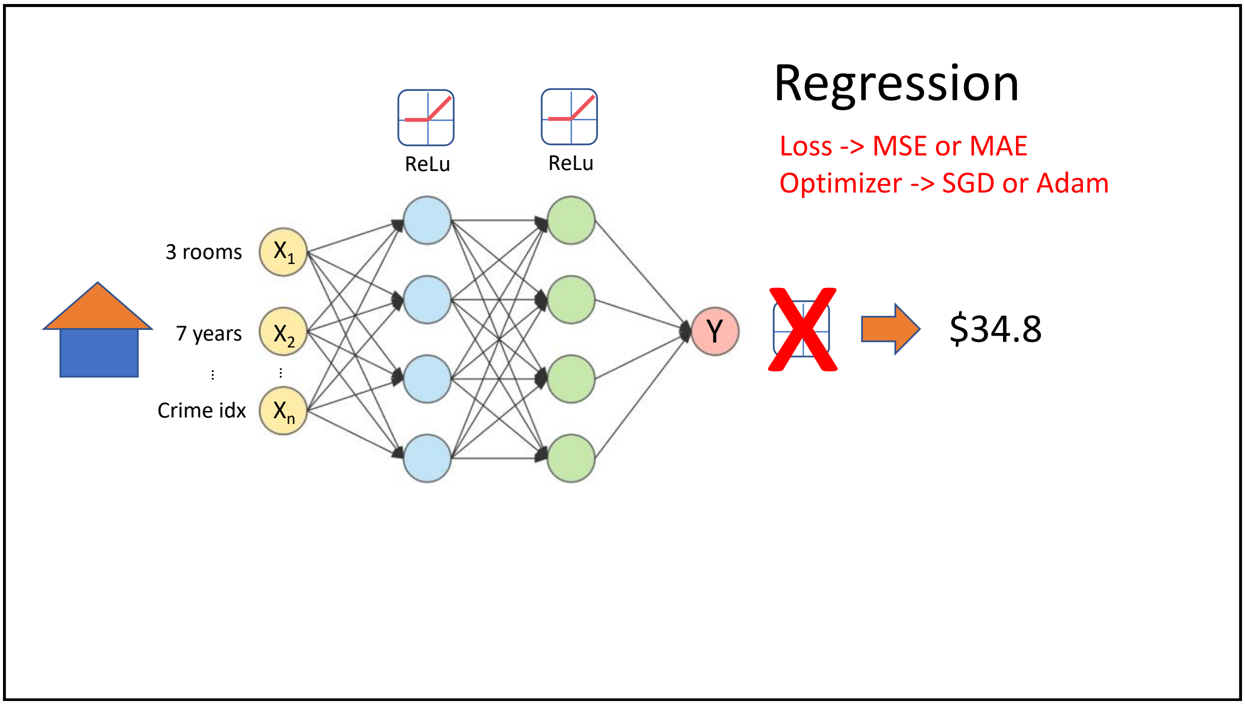
1

Loss, Optimizer

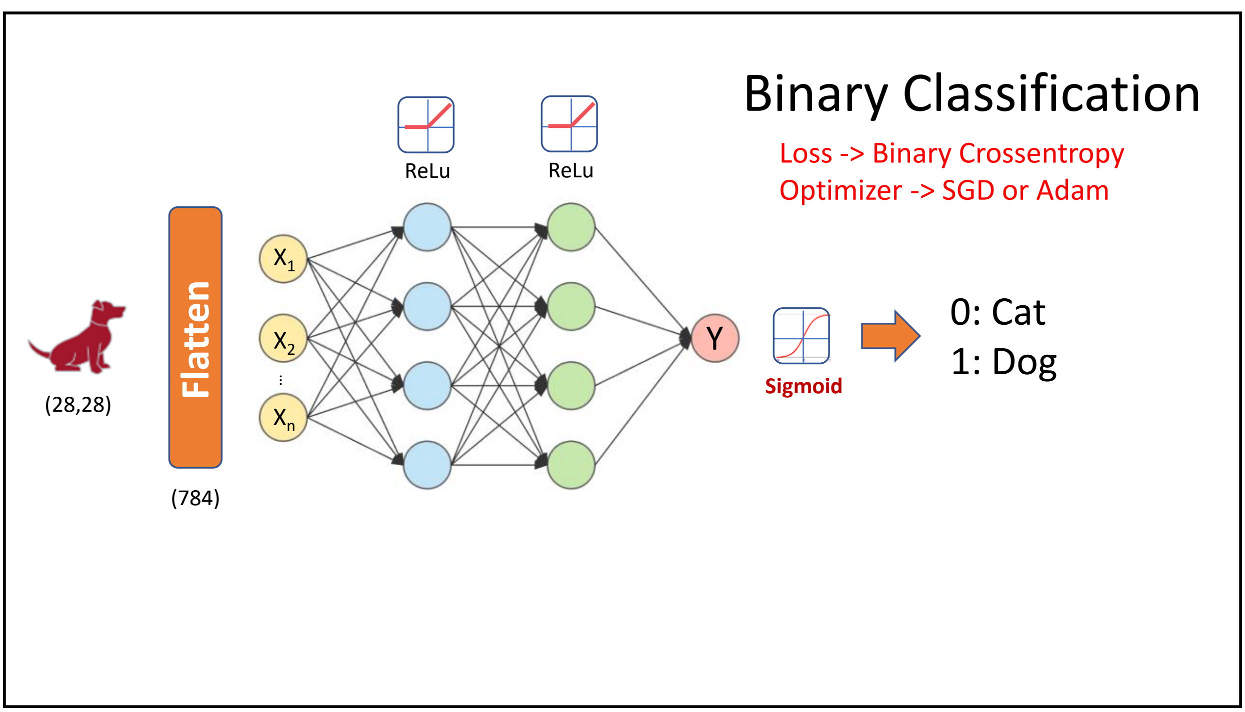
Quick Recap

2

2


Regression
Loss -> MSE or MAE
Optimizer -> SGD or Adam
ReLu
ReLu
3 rooms
7 years
Crime idx
X1
X2
Xn
Y
$34.8

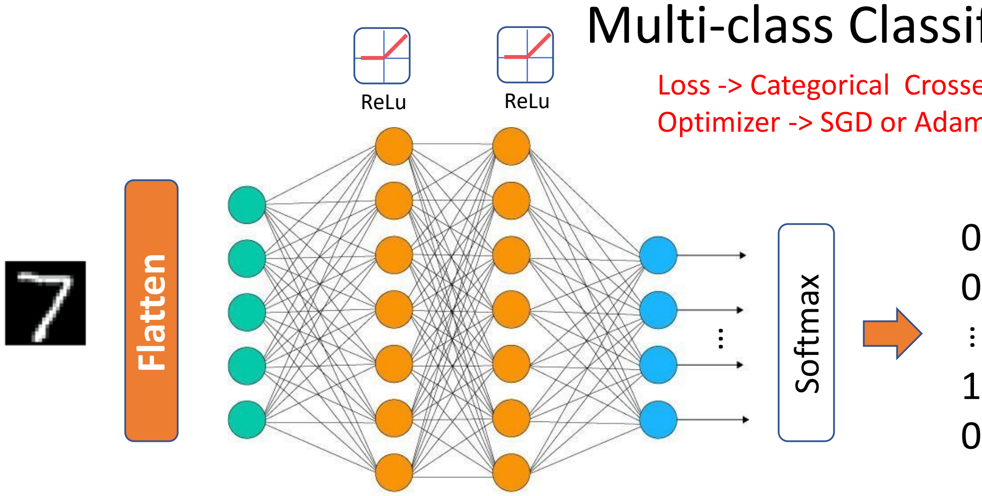
3


Binary Classification
Loss -> Binary Crossentropy
Optimizer -> SGD or Adam
ReLu
ReLu
(28,28)
Flatten
(784)
X1
X2
Xn
Y
Sigmoid
0: Cat
1: Dog

4

Multi-class Classification

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



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

5

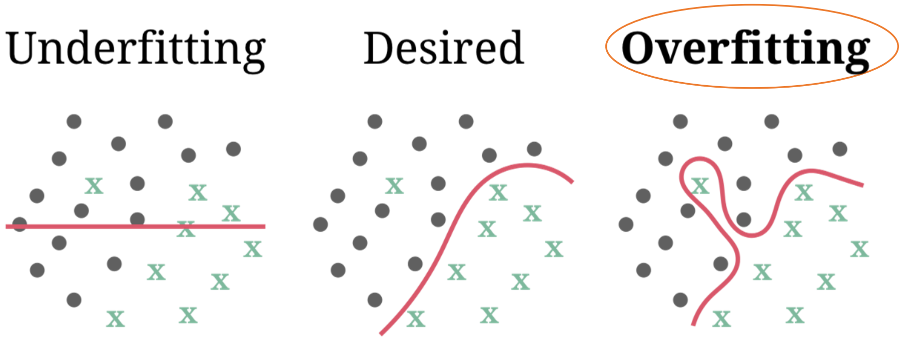
Train, Validate, Test

Importance of Data

6

Data

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



7

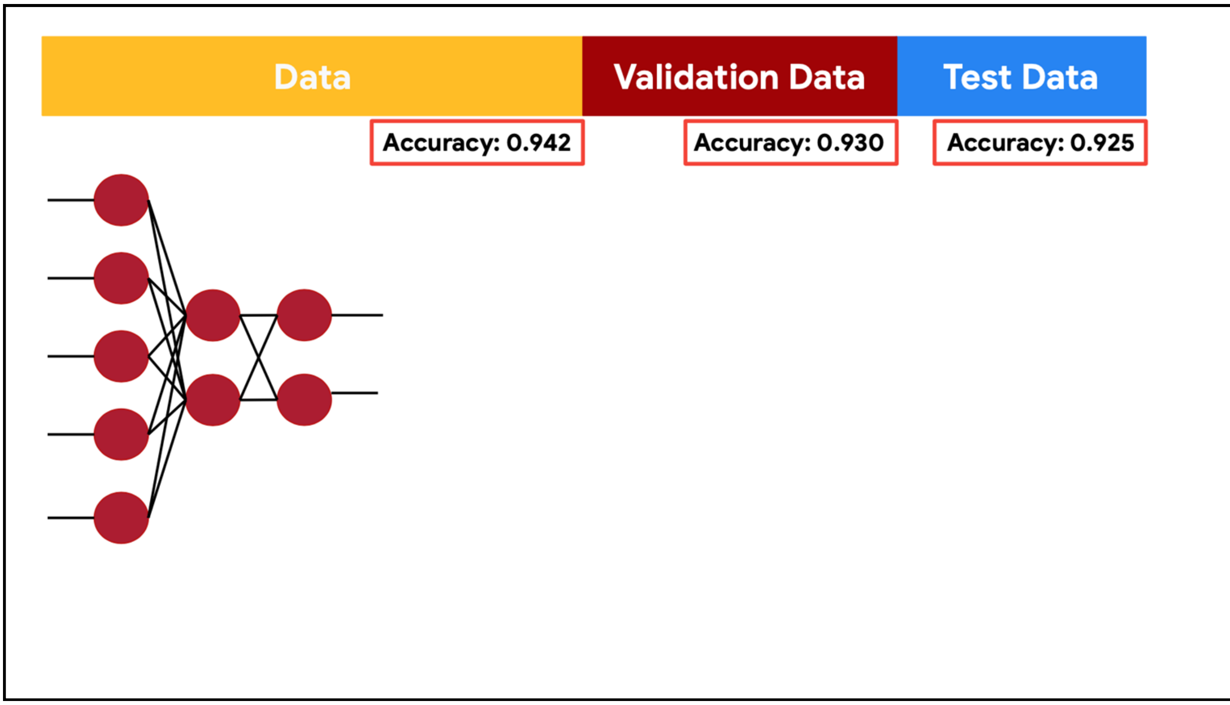
Data	Validation Data	Test Data

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).

Used to evaluate the current training epoch

Used to evaluate the final model after training

8



9

Digits Classification: Validation and Test, Learning Rate

Code Time!

TF_MNIST_Classification_v2.ipynb

10

10

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

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

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

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

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

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

11

```
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,)
```

12

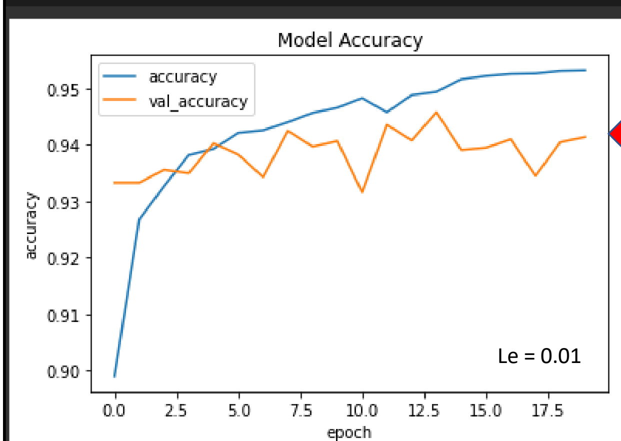
```python
history = model.fit(
      train_images,
      train_labels,
      epochs=20,
      validation_data=(val_images, val_labels)
      )
```

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 on its own.

13

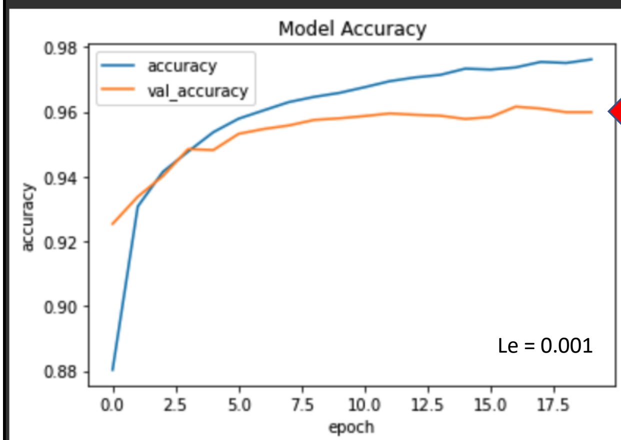
```python
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.title('Model Accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(loc='upper left')
plt.show()
```



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

14

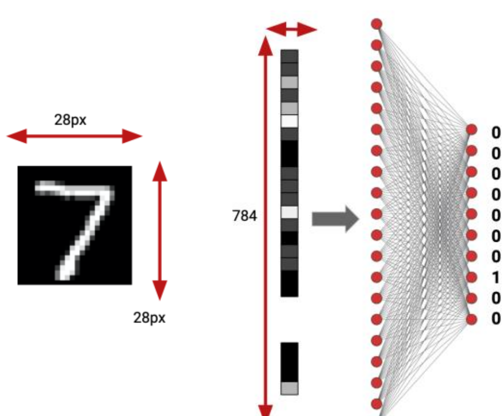
```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.title('Model Accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(loc='upper left')
plt.show()
```



If validation accuracy goes down (or becomes stable), even if train accuracy goes up, it means that probably the model is overfitting. In this case the training process should terminate – and should not continue with more epochs.

15

Data	Validation Data	Test Data
Accuracy: 0.976	Accuracy: 0.963	Accuracy: 0.957



16

In Summary, Remember:

- **Training Data**
  - Used to train model **parameters**
- **Validation Data**
  - Used to determine what model **hyperparameters** to adjust (and re-train)
- **Test Data**
  - Used to compute model final performance metrics

17

Model Performance Metrics

Classification

18

18

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]

19

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**

20

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)**

[Source: https://en.wikipedia.org/wiki/Confusion_matrix]

21

Type I error (false positive)

Type II error (false negative)

22

Precision vs. Accuracy

In a set of measurements:

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

High Precision, High Accuracy

Low Precision, High Accuracy

High Precision, Low Accuracy

Low Precision, Low Accuracy

23

Accuracy, Precision and Recall

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

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

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

(or Sensitivity)

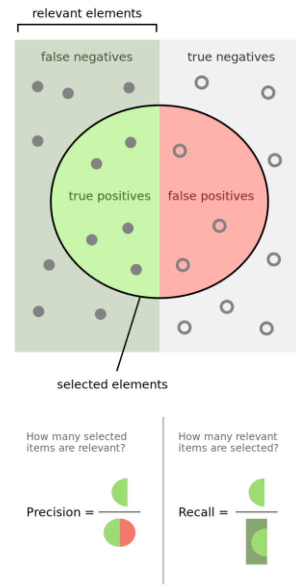
24

F1-Score

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

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

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



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

25

```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
```

26

Classification Report: Confusion Matrix

Code Time!

Classification_Report.ipynb

27

27

Credits

- A previous edition of this course was developed in collaboration with Dr. Susan C. Schneider of Marquette University.
- We are very grateful and thank all the following professors, researchers, and practitioners for jump-starting courses on TinyML and for sharing their teaching materials:
- Prof. Marcelo Rovai - TinyML - Machine Learning for Embedding Devices, UNIFEI
  - https://github.com/Mjrovai/UNIFEI-IESTI01-TinyML-2022.1
- Prof. Vijay Janapa Reddi - CS249r: Tiny Machine Learning, Applied Machine Learning on Embedded IoT Devices, Harvard
  - https://sites.google.com/g.harvard.edu/tinyml/home
- Prof. Rahul Mangharam – ESE3600: Tiny Machine Learning, Univ. of Pennsylvania
  - https://tinyml.seas.upenn.edu/#
- Prof. Brian Plancher - Harvard CS249r: Tiny Machine Learning (TinyML), Barnard College, Columbia University
  - https://a2r-lab.org/courses/cs249r_tinyml/

28

28

References

- Additional references from where information and other teaching materials were gathered include:

- Applications & Deploy textbook: "TinyML" by Pete Warden, Daniel Situnayake
  - https://www.oreilly.com/library/view/tinyml/9781492052036/
- Deploy textbook "TinyML Cookbook" by Gian Marco Iodice
  - https://github.com/PacktPublishing/TinyML-Cookbook
- Jason Brownlee
  - https://machinelearningmastery.com/
- TinyMLedu
  - https://tinyml.seas.harvard.edu/
- Professional Certificate in Tiny Machine Learning (TinyML) – edX/Harvard
  - https://www.edx.org/professional-certificate/harvardx-tiny-machine-learning
- Introduction to Embedded Machine Learning - Coursera/Edge Impulse
  - https://www.coursera.org/learn/introduction-to-embedded-machine-learning
- Computer Vision with Embedded Machine Learning - Coursera/Edge Impulse
  - https://www.coursera.org/learn/computer-vision-with-embedded-machine-learning