

TinyML 도서 스터디 2주차



Chapter 4 ~ Chapter 5

이름

신정아

E-mail

jeongah.arie@gmail.com

오늘의 이야기는 ...



Chapter 4 ~ Chapter 5

Chapter 4. The “Hello World” of TinyML: Building and Training a Model

Chapter 5. The “Hello World” of TinyML: Building an Application

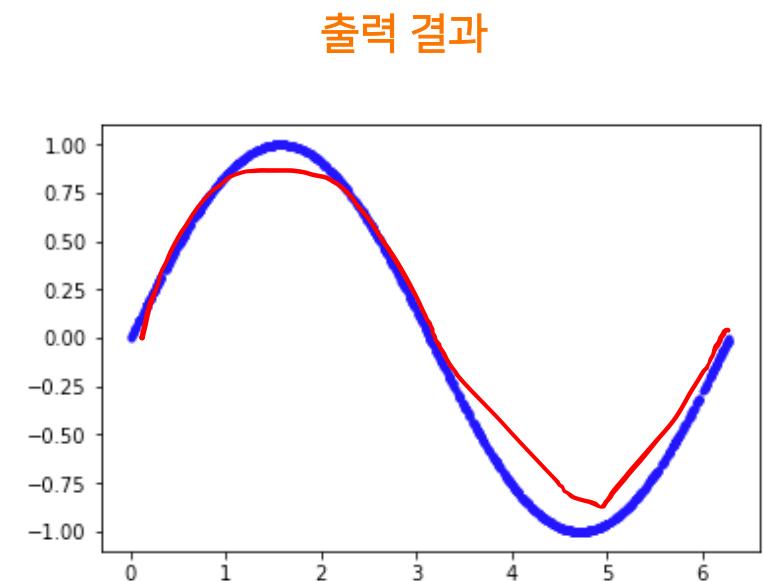
Chapter 4. The “Hello World” of TinyML : Building and Training a Model

Generating Data

Chapter 4. The “Hello World” of TinyML : Building and Training a Model

난수 생성 할 때마다 값이 달라지는 것이 아니라, 누가, 언제 하든지 간에 똑같은 난수 생성을 원한다면 (reproducibility 확보) seed 번호를 지정

```
# 아래의 값만큼 데이터 샘플을 생성할 것이다.  
SAMPLES = 1000  
# 시드 값을 지정하여 이 노트북에서 실행할 때마다 다른 랜덤 값을 얻게 한다.  
# 어떤 숫자든 사용할 수 있다  
np.random.seed(1337)  
# 사인파 진폭의 범위인 0~2π 내에서 균일하게 분포된 난수 집합을 생성한다.  
x_values = np.random.uniform(low=0, high=2*math.pi, size=SAMPLES)  
# 값을 섞어서 생성된 값들이 순서를 따르지 않도록 한다.  
np.random.shuffle(x_values)  
# 해당 사인값을 계산한다  
y_values = np.sin(x_values)  
# 데이터를 그래프로 그린다. 'b.' 인수는 라이브러리에 점을 파란색으로 출력하도록 지시한다.  
plt.plot(x_values, y_values, 'b.')  
plt.show()
```



Adding Noise to data

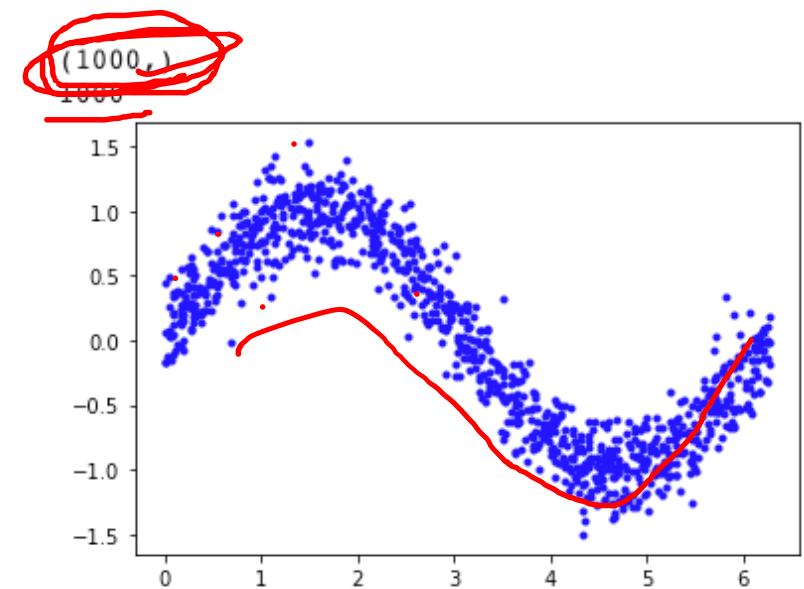
Chapter 4. The “Hello World” of TinyML : Building and Training a Model

For more reflective of a real-world situation (quite messy data)

`np.random.randn(*x.shape)` ↗ x의 shape을 가진 표준 가우시안 정규분포

```
● ● ●  
  
print(y_values.shape)  
print(*y_values.shape)  
  
# 각 y 값에 임의의 작은 숫자를 추가한다.  
y_values += 0.1 * np.random.randn(*y_values.shape)  
  
# 그래프를 생성한다.  
plt.plot(x_values, y_values, 'b.')  
plt.show()
```

출력 결과



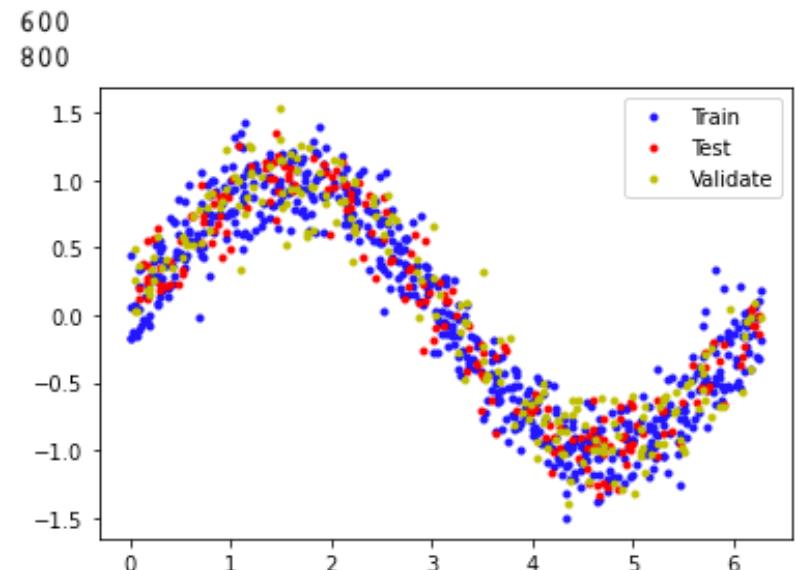
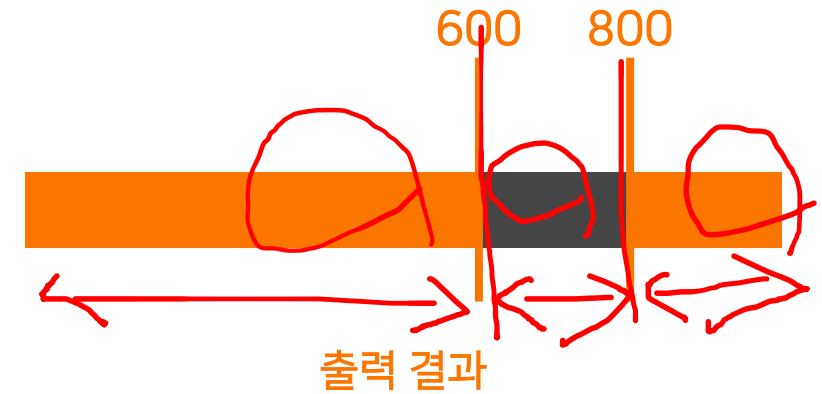
Splitting the data

Chapter 4. The “Hello World” of TinyML : Building and Training a Model

20% for validation, 20% for testing and 60% for training a model



```
# 훈련에 60%, 테스트에 20%, 검증에 20%를 사용한다.  
# 각 항목의 인덱스를 계산한다.  
TRAIN_SPLIT = int(0.6 * SAMPLES)  
TEST_SPLIT = int(0.2 * SAMPLES + TRAIN_SPLIT)  
  
print(TRAIN_SPLIT)  
print(TEST_SPLIT)  
  
# np.split을 사용하여 데이터를 세 부분으로 자른다.  
# np.split의 두 번째 인수는 데이터가 분할되는 인덱스 배열이며,  
# 우리는 두 개의 인덱스를 제공하므로 데이터는 세 개의 덩어리로 나뉠 것이다.  
x_train, x_test, x_validate = np.split(x_values, [TRAIN_SPLIT, TEST_SPLIT])  
y_train, y_test, y_validate = np.split(y_values, [TRAIN_SPLIT, TEST_SPLIT])  
  
# 분할한 데이터를 합쳤을 때 원래의 사이즈와 같은지 재확인한다.  
assert (x_train.size + x_validate.size + x_test.size) == SAMPLES  
  
# 분할된 각 데이터들을 다른 색상으로 그래프에 표시한다.  
plt.plot(x_train, y_train, 'b.', label="Train")  
plt.plot(x_test, y_test, 'r.', label="Test")  
plt.plot(x_validate, y_validate, 'y.', label="Validate")  
plt.legend()  
plt.show()
```



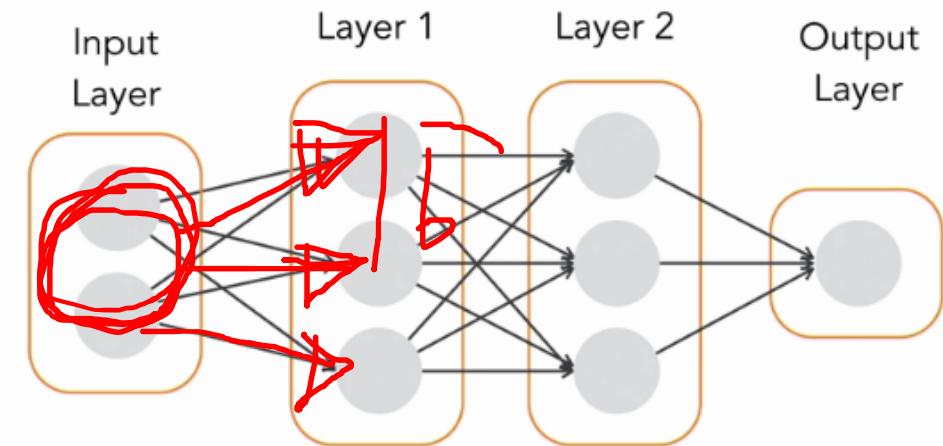
Defining a Basic Model

Chapter 4. The “Hello World” of TinyML : Building and Training a Model

Input: x, Output: sin x

```
# 간단한 모델 구조를 만들기 위해 케라스를 사용한다.  
from tensorflow.keras import layers  
model_1 = tf.keras.Sequential()  
  
# 첫 번째 레이어는 16 개의 뉴런을 통해 스칼라Scalar 입력을 받아 다음 레이어에 전달한다.  
# 뉴런은 'relu' 활성화 함수에 따라 값을 전달할지 말지를 결정하게 된다.  
model_1.add(layers.Dense(16, activation='relu', input_shape=(1,)))  
  
# 마지막 레이어는 하나의 뉴런인데, 원하는 결과값이 하나의 값이기 때문이다.  
model_1.add(layers.Dense(1))  
  
# 표준 옵티마이저와 손실을 사용하여 회귀 모델을 컴파일한다.  
model_1.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])
```

Sequential Model



단일 입력값 x와 16개의 뉴런
Dense Layer (Fully Connected Layer)
👉 입력이 모든 단일 뉴런으로 전달됨

Defining a Basic Model

Chapter 4. The “Hello World” of TinyML : Building and Training a Model

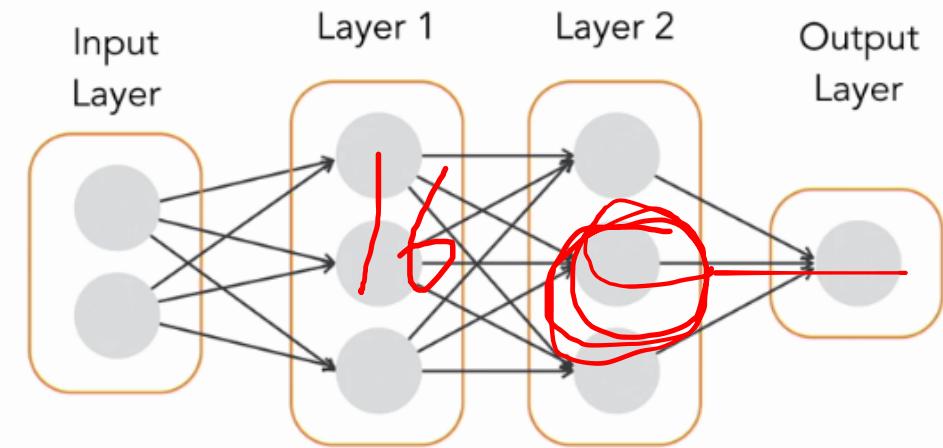
Input: x, Output: $\sin x$



```
# 간단한 모델 구조를 만들기 위해 케라스를 사용한다.  
from tensorflow.keras import layers  
model_1 = tf.keras.Sequential()  
  
# 첫 번째 레이어는 16 개의 뉴런을 통해 스칼라Scalar 입력을 받아 다음 레이어에 전달한다.  
# 뉴런은 'relu' 활성화 함수에 따라 값을 전달할지 말지를 결정하게 된다.  
model_1.add(layers.Dense(16, activation='relu', input_shape=(1,)))  
  
# 마지막 레이어는 하나의 뉴런인데, 원하는 결과값이 하나의 값이기 때문이다.  
model_1.add(layers.Dense(1))  
  
# 표준 옵티마이저와 손실을 사용하여 회귀 모델을 컴파일한다.  
model_1.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])
```

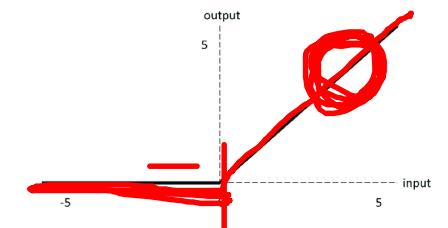
Reference

Sequential Model



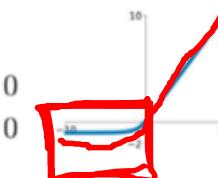
ReLU Function

```
def relu(input):  
    return max(0.0, input)
```

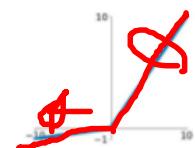


ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Leaky ReLU
 $\max(0.1x, x)$



Defining a Basic Model

Chapter 4. The “Hello World” of TinyML : Building and Training a Model

Input: x, Output: sin x



```
# 간단한 모델 구조를 만들기 위해 케라스를 사용한다.  
from tensorflow.keras import layers  
model_1 = tf.keras.Sequential()  
  
# 첫 번째 레이어는 16 개의 뉴런을 통해 스칼라Scalar 입력을 받아 다음 레이어에 전달한다.  
# 뉴런은 'relu' 활성화 함수에 따라 값을 전달할지 말지를 결정하게 된다.  
model_1.add(layers.Dense(16, activation='relu', input_shape=(1,)))  
  
# 마지막 레이어는 하나의 뉴런인데, 원하는 결과값이 하나의 값이기 때문이다.  
model_1.add(layers.Dense(1))  
  
# 표준 옵티마이저와 손실을 사용하여 회귀 모델을 컴파일한다.  
model_1.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])
```

Optimizer - rmsprop

$$\overline{h_i} \leftarrow \rho \overline{h_{i-1}} + (1 - \rho) \frac{\partial L_i}{\partial W} \odot \frac{\partial L_i}{\partial W}$$

RMSProp 은 기울기를 단순 누적하지 않고 지수 가중 이동 평균 Exponentially weighted moving average 를 사용하여 최신 기울기들이 더 크게 반영 (compared w/ AdaGrad)

Loss - mse (Mean Squared Error)

$$\frac{1}{n} \sum_{i=1}^n (y_i - t_i)^2$$

y_i 는 신경망의 출력,
 t_i 는 정답 레이블(One-Hot 인코딩 - 즉 정답만 1로 표시 나머진 0)

Defining a Basic Model

Chapter 4. The “Hello World” of TinyML : Building and Training a Model

Input: x, Output: $\sin x$



```
# 간단한 모델 구조를 만들기 위해 케라스를 사용한다.
```

```
from tensorflow.keras import layers  
model_1 = tf.keras.Sequential()
```

```
# 첫 번째 레이어는 16 개의 뉴런을 통해 스칼라Scalar 입력을 받아 다음 레이어에 전달한다.
```

```
# 뉴런은 'relu' 활성화 함수에 따라 값을 전달할지 말지를 결정하게 된다.
```

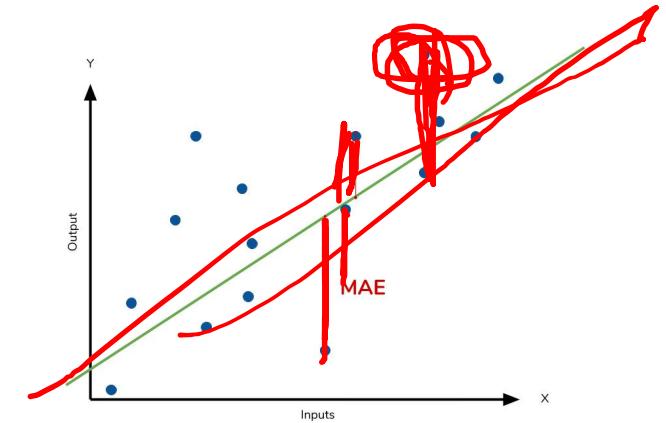
```
model_1.add(layers.Dense(16, activation='relu', input_shape=(1,)))
```

```
# 마지막 레이어는 하나의 뉴런인데, 원하는 결과값이 하나의 값이기 때문이다.
```

```
model_1.add(layers.Dense(1))
```

```
# 표준 옵티마이저와 손실을 사용하여 회귀 모델을 컴파일한다.
```

```
model_1.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])
```



Metrics for judging performance – mae (Mean Absolute Error)

$$MAE = \frac{\sum |y - \hat{y}|}{n} \quad \text{모델의 예측값과 실제값의 차이를 모두 더한다}$$

- 절대값을 취하기 때문에 가장 직관적으로 알 수 있는 지표
- MSE 보다 특이치에 robust 함.
- 절대값을 취하기 때문에 모델이 underperformance 인지 overperformance 인지 알 수 없음.
 - underperformance: 모델이 실제보다 낮은 값으로 예측
 - overperformance: 모델이 실제보다 높은 값으로 예측

Training our model

Chapter 4. The “Hello World” of TinyML : Building and Training a Model

```
# 모델 훈련
history_1 = model_1.fit(x_train, y_train, epochs=1000, batch_size=16,
                        validation_data=(x_validate, y_validate))
                    )  
[call]
```

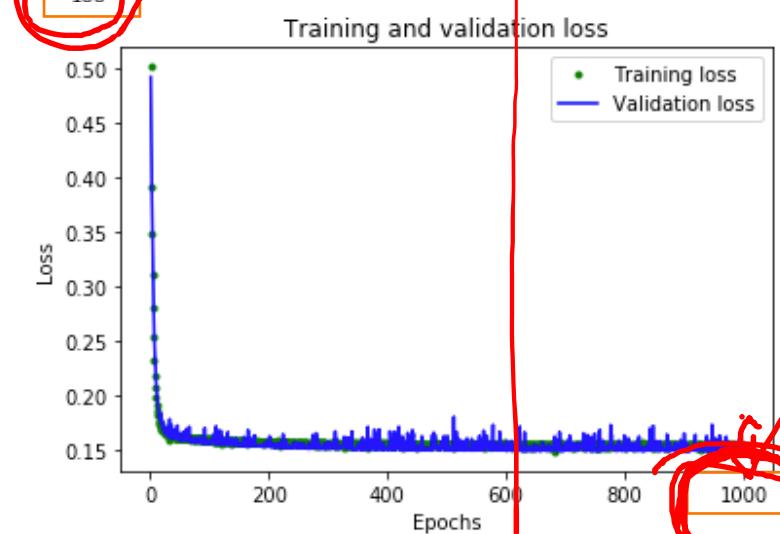
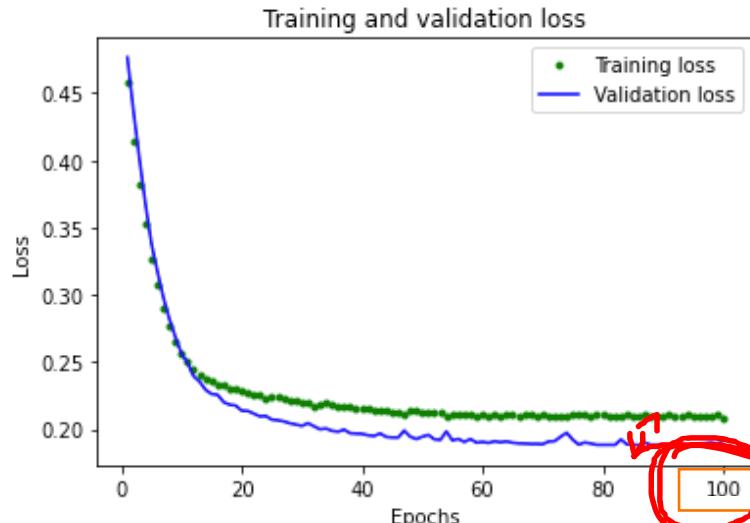
```
Train on 600 samples, validate on 200 samples
Epoch 1/1000
600/600 [=====] - 0s 412us/sample - loss: 0.5016 - mae: 0.6297 - val_loss: 0.4922 - val_mae: 0.6235
Epoch 2/1000
600/600 [=====] - 0s 105us/sample - loss: 0.3905 - mae: 0.5436 - val_loss: 0.4262 - val_mae: 0.5641
...
Epoch 998/1000
600/600 [=====] - 0s 109us/sample - loss: 0.1535 - mae: 0.3068 - val_loss: 0.1507 - val_mae: 0.3113
Epoch 999/1000
600/600 [=====] - 0s 100us/sample - loss: 0.1545 - mae: 0.3077 - val_loss: 0.1499 - val_mae: 0.3103
Epoch 1000/1000
600/600 [=====] - 0s 132us/sample - loss: 0.1530 - mae: 0.3045 - val_loss: 0.1542 - val_mae: 0.3143
```

Graphing the history

Chapter 4. The “Hello World” of TinyML : Building and Training a Model

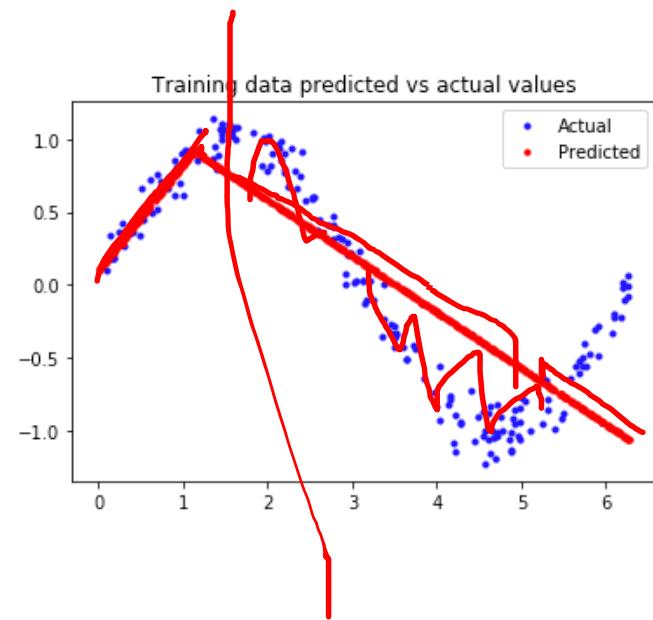
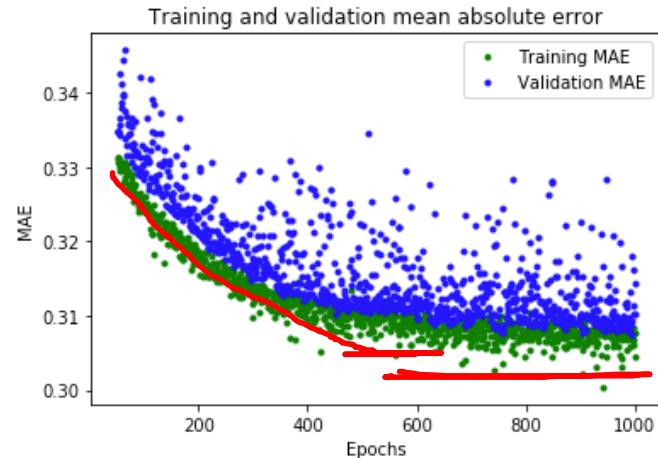
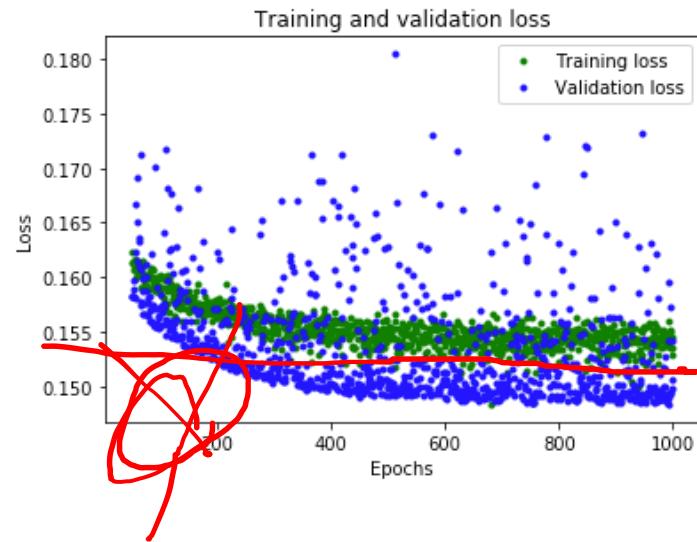


```
# 훈련과 검증 과정에서 예측값과 실제값 사이의 거리인 손실 그래프를 그린다.  
loss = history_1.history['loss']  
val_loss = history_1.history['val_loss']  
  
epochs = range(1, len(loss) + 1)  
  
plt.plot(epochs, loss, 'g.', label='Training loss')  
plt.plot(epochs, val_loss, 'b', label='Validation loss')  
plt.title('Training and validation loss')  
plt.xlabel('Epochs')  
plt.ylabel('Loss')  
plt.legend()  
plt.show()
```



Graphing the history

Chapter 4. The “Hello World” of TinyML : Building and Training a Model



Improving our model

Chapter 4. The “Hello World” of TinyML : Building and Training a Model

```
model_2 = tf.keras.Sequential()

# 첫 번째 레이어는 스칼라 입력을 받아 16개의 뉴런을 통해 전달하고,
# 뉴런은 'relu' 활성화 함수에 따라 활성화 여부를 결정한다.
model_2.add(layers.Dense(16, activation='relu', input_shape=(1,)))

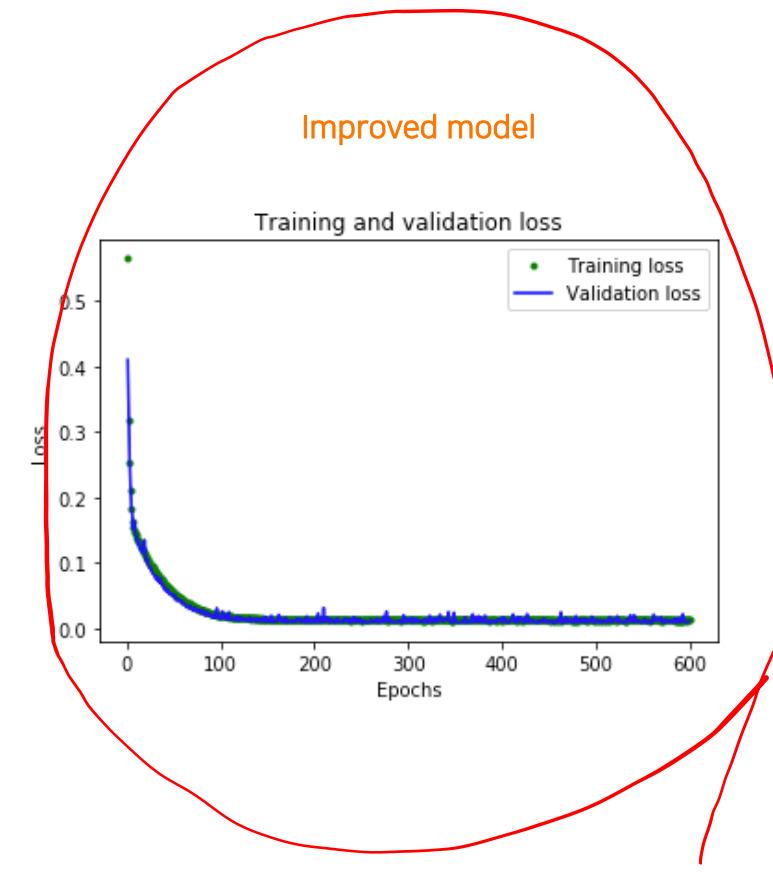
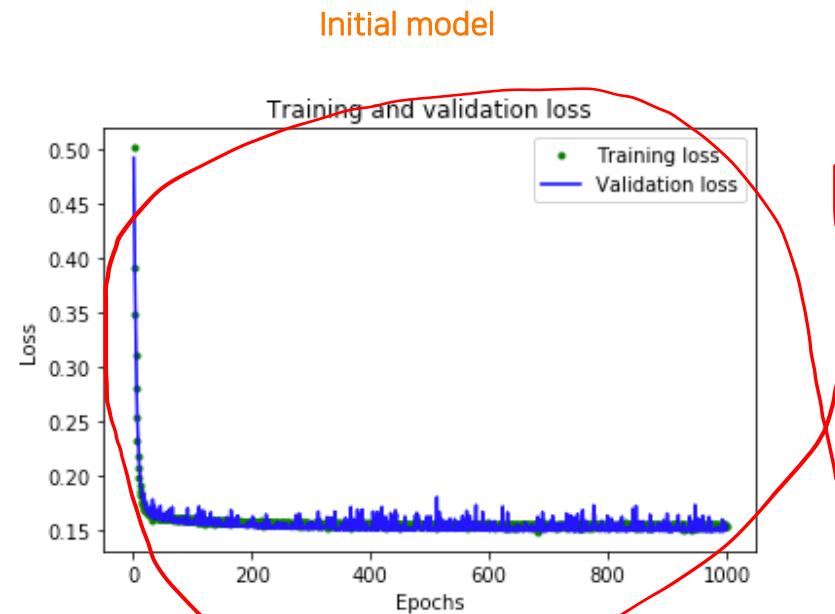
# 새로운 두 번째 레이어는 네트워크가 더 복잡한 표현을 배우는 데 도움을 준다.
model_2.add(layers.Dense(16, activation='relu')) # 16

# 단일 값을 출력해야 하기 때문에 최종 레이어는 단일 뉴런으로 구성된다.
model_2.add(layers.Dense(1))

# 표준 옵티마이저 및 손실 함수를 사용하여 회귀 모델을 컴파일
model_2.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])
```

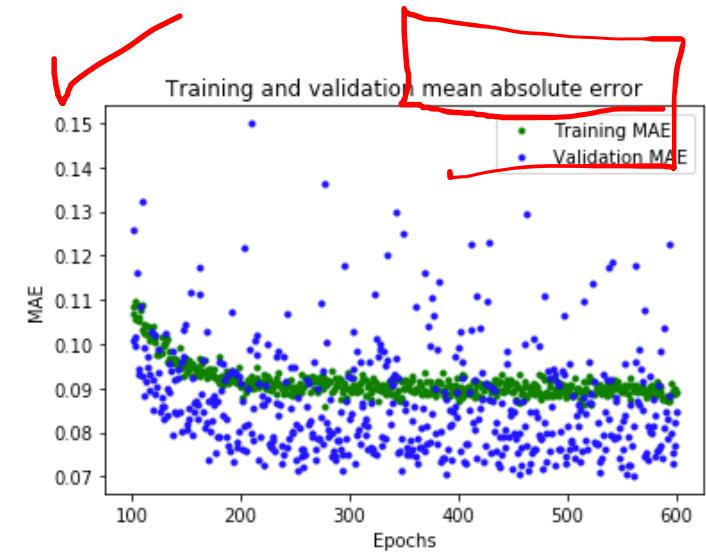
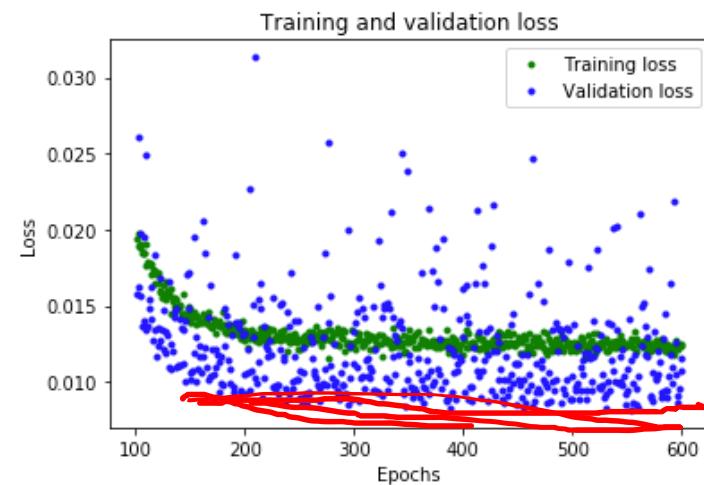
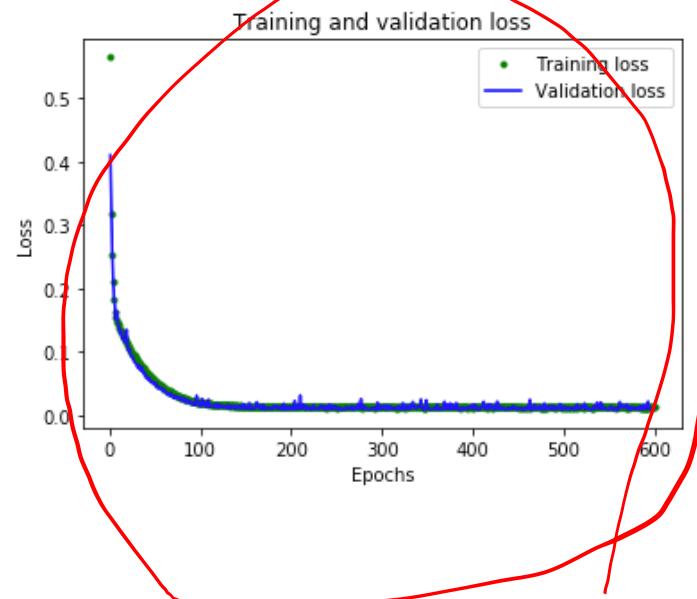
Improving our model

Chapter 4. The “Hello World” of TinyML : Building and Training a Model



Improving our model

Chapter 4. The “Hello World” of TinyML : Building and Training a Model



Converting the model for TFLite

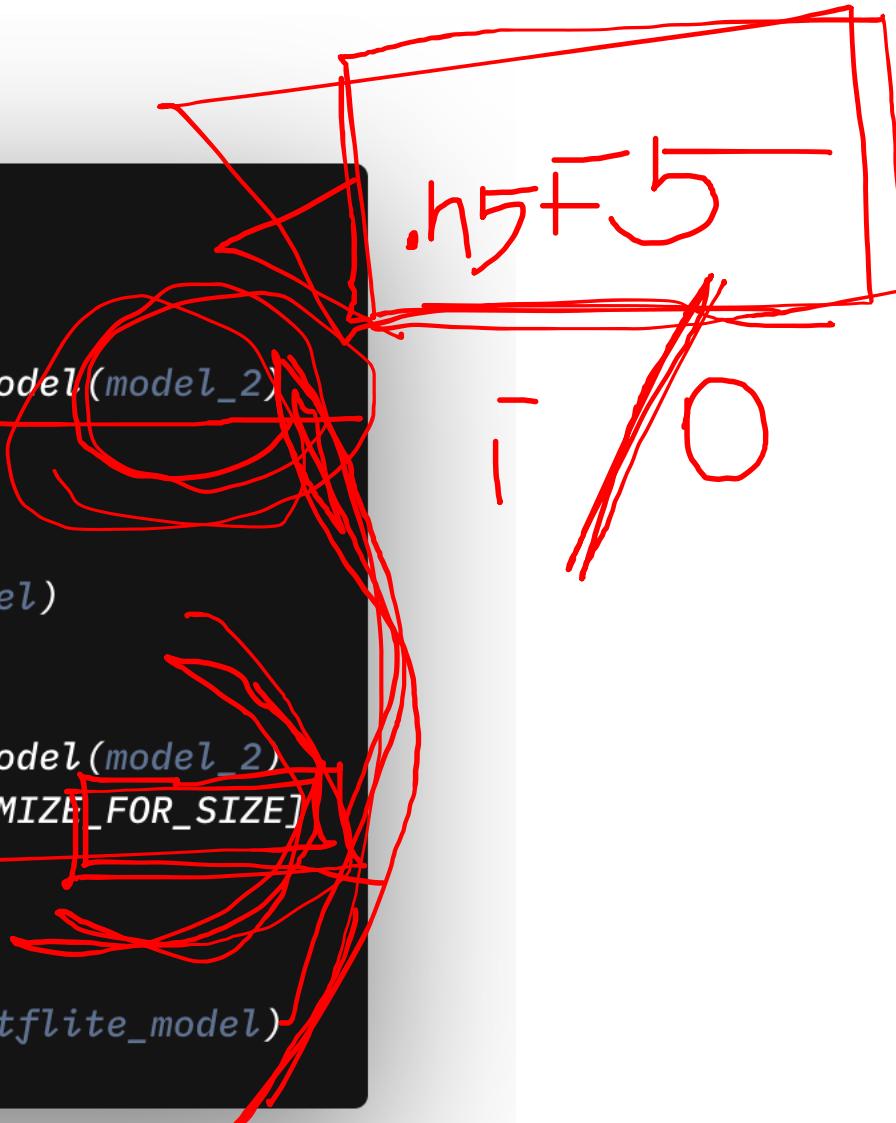
Chapter 4. The “Hello World” of TinyML : Building and Training a Model

```
# 양자화 없이 모델을 텐서플로우 라이트 형식으로 변환
converter = tf.lite.TFLiteConverter.from_keras_model(model_2)
tflite_model = converter.convert()

# 모델을 디스크에 저장
open("sine_model.tflite", "wb").write(tflite_model)

# 양자화하여 모델을 텐서플로우 라이트 형식으로 변환
converter = tf.lite.TFLiteConverter.from_keras_model(model_2)
converter.optimizations = [tf.lite.Optimize.OPTIMIZE_FOR_SIZE]
tflite_model = converter.convert()

# 모델을 디스크에 저장
open("sine_model_quantized.tflite", "wb").write(tflite_model)
```



Converting the model for TFlite

Chapter 4. The “Hello World” of TinyML : Building and Training a Model



```
# 양자화 없이 모델을 텐서플로우 라이트 형식으로 변환
converter = tf.lite.TFLiteConverter.from_keras_model(model_2)
tflite_model = converter.convert()

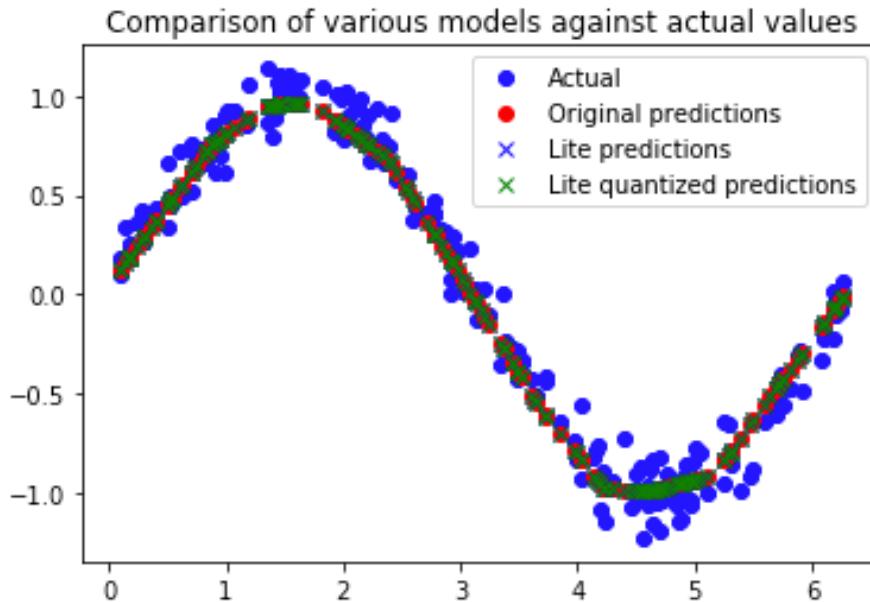
# 모델을 디스크에 저장
open("sine_model.tflite", "wb").write(tflite_model)

# 양자화하여 모델을 텐서플로우 라이트 형식으로 변환
converter = tf.lite.TFLiteConverter.from_keras_model(model_2)
converter.optimizations = [tf.lite.Optimize.OPTIMIZE_FOR_SIZE]
tflite_model = converter.convert()

# 모델을 디스크에 저장
open("sine_model_quantized.tflite", "wb").write(tflite_model)
```

Converting the model for TFlite

Chapter 4. The “Hello World” of TinyML : Building and Training a Model



```
# 각 모델에 대한 인터프리터 생성
sine_model = tf.lite.Interpreter('sine_model.tflite')
sine_model_quantized = tf.lite.Interpreter('sine_model_quantized.tflite')

# 각 모델에 대한 메모리 할당
sine_model.allocate_tensors()
sine_model_quantized.allocate_tensors()

# 입력과 결과 텐서에 대한 인덱스 가져오기
sine_model_input = sine_model.tensor(sine_model.get_input_details()[0]["index"])
sine_model_output = sine_model.tensor(sine_model.get_output_details()[0]["index"])
sine_model_quantized_input = sine_model_quantized.tensor(sine_model_quantized.get_input_details()[0]
["index"])
sine_model_quantized_output = sine_model_quantized.tensor(sine_model_quantized.get_output_details()[0]
["index"])

# 결과를 저장하기 위한 배열 생성
sine_model_predictions = np.empty(x_test.size)
sine_model_quantized_predictions = np.empty(x_test.size)

# 각 값에 대해 각 모델의 인터프리터를 실행하고 결과를 배열에 저장
for i in range(x_test.size):
    sine_model_input().fill(x_test[i])
    sine_model.invoke()
    sine_model_predictions[i] = sine_model_output()[0]

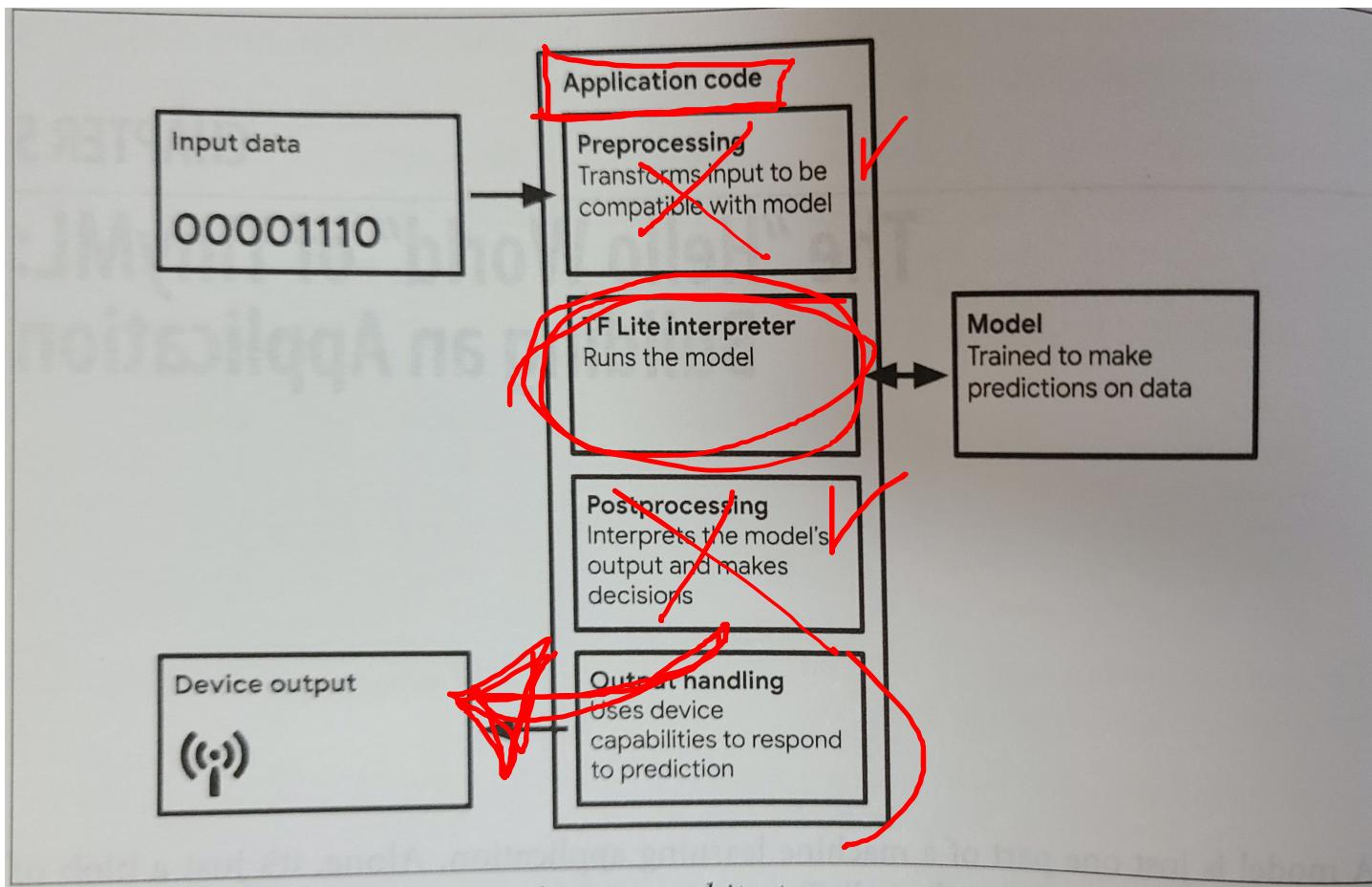
    sine_model_quantized_input().fill(x_test[i])
    sine_model_quantized.invoke()
    sine_model_quantized_predictions[i] = sine_model_quantized_output()[0]

# 데이터가 어떻게 정렬되는지 확인
plt.clf()
plt.title('Comparison of various models against actual values')
plt.plot(x_test, y_test, 'bo', label='Actual')
plt.plot(x_test, predictions, 'ro', label='Original predictions')
plt.plot(x_test, sine_model_predictions, 'bx', label='Lite predictions')
plt.plot(x_test, sine_model_quantized_predictions, 'gx', label='Lite quantized predictions')
plt.legend()
plt.show()
```

Chapter 5. The “Hello World” of TinyML: Building an Application

TinyML Application Architecture

Chapter 5. The “Hello World” of TinyML: Building an Application



Android implementation

Chapter 5. The “Hello World” of TinyML: Building an Application

A screenshot of an Android application's code editor and a floating "MODEL PROPERTIES" dialog. The code shows a try-with-resources block for an Interpreter. The "dense_2_input" tensor and the "Identity" output tensor are circled in red.

```
try (Interpreter interpreter = new Interpreter(file_of_a_tensorflowlite_model)) {
    interpreter.run(input, output);
}
```

