

자동차 연비 (auto mpg: mile per gallon) 데이터로 회귀분석

<https://www.tensorflow.org/tutorials/keras/regression?hl=ko>

회귀와 분류

- **회귀(regression)**
 - 가격이나 확률 같이 연속된 출력 값을 예측하는 것이 목적
- **분류(classification)**
 - 여러 개의 클래스 중 하나의 클래스를 선택하는 것이 목적
 - 예를 들어, 사진에 사과 또는 오렌지가 포함되어 있을 때 어떤 과일인지 인식하는 것
- **자동차 연비를 예측하는 모델**
 - Auto MPG 데이터 셋을 사용
 - 1970년대 후반과 1980년대 초반의 데이터
 - 이 기간에 출시된 자동차 정보를 모델에 제공
 - 실린더 수, 배기량, 마력(horsepower), 공차 중량 같은 속성
 - 이 예제는 tf.keras API를 사용
- **Mpg**
 - Mile per gallon
 - 연비
 - km/l

파일

- `reg_mpg_tuto.ipynb`

seaborn 설치와 모듈 가져 오기

- seaborn 설치

- !pip install seaborn
- !pip install -q seaborn
 - 메시지 출력을 최소화
- pip install seaborn

- 모듈 가져 오기

필요 모듈 가져오기

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

pip 명령

pip 옵션 및 명령

pip의 기능을 살펴보기 위해 `pip --help`를 통해 살펴보면 다음과 같다.

Usage:

`pip <command> [options]`

Commands:

<code>install</code>	Install packages.
<code>download</code>	Download packages.
<code>uninstall</code>	Uninstall packages.
<code>freeze</code>	Output installed packages in <code>requirements format</code> .
<code>list</code>	List installed packages.
<code>show</code>	Show information about installed packages.
<code>check</code>	Verify installed packages have compatible dependencies.
<code>search</code>	Search PyPI for packages.
<code>wheel</code>	Build wheels from your requirements.
<code>hash</code>	Compute hashes of package archives.
<code>completion</code>	A helper command used for command completion.
<code>help</code>	Show help for commands.

Auto MPG 데이터셋

- UCI 머신 러닝 저장소에서 다운로드

```
# 데이터 가져오기
dataset_path = keras.utils.get_file("auto-mpg.data",
                                     "http://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.data")
print(dataset_path)
```

- 판다스를 사용하여 데이터를 읽기

```
# 데이터 읽어 dataset에 저장
col_names = ['MPG', 'Cylinders', 'Displacement', 'Horsepower', 'Weight', 'Acceleration',
             'Model Year', 'Origin']
raw_data = pd.read_csv(dataset_path,
                       names=col_names, na_values = "?", comment='\t', sep=" ",
                       skipinitialspace=True)
dataset = raw_data.copy()
dataset.tail(10)
```

특정 문자가 있는 행은 주석으로 간주하고 읽지 않고 건너뛴

데이터 셋

```
In [31]: 1 dataset_path = keras.utils.get_file("auto-mpg.data", "http://archive.ics.uci.edu/n
2 print(dataset_path)
3
4 # 데이터 읽어 dataset에 저장
5 column_names = ['MPG', 'Cylinders', 'Displacement', 'Horsepower', 'Weight',
6                 'Acceleration', 'Model Year', 'Origin']
7 raw_dataset = pd.read_csv(dataset_path, names=column_names, na_values = "?",
8                           comment='#t', sep=" ", skipinitialspace=True)
9 dataset = raw_dataset.copy()
10 dataset.head(10)
```

C:\Users\USER\keras\datasets\auto-mpg.data

Out [31]:


	MPG	Cylinders	Displacement	Horsepower	Weight	Acceleration	Model Year	Origin
0	18.0	8	307.0	130.0	3504.0	12.0	70	1
1	15.0	8	350.0	165.0	3693.0	11.5	70	1
2	18.0	8	318.0	150.0	3436.0	11.0	70	1
3	16.0	8	304.0	150.0	3433.0	12.0	70	1
4	17.0	8	302.0	140.0	3449.0	10.5	70	1
5	15.0	8	429.0	198.0	4341.0	10.0	70	1
6	14.0	8	454.0	220.0	4354.0	9.0	70	1
7	14.0	8	440.0	215.0	4312.0	8.5	70	1
8	14.0	8	455.0	225.0	4425.0	10.0	70	1
9	15.0	8	390.0	190.0	3850.0	8.5	70	1

속성과 데이터 소스 사이트

- **Attribute Information:**
- 1. mpg: continuous
- 2. cylinders: multi-valued discrete
- 3. displacement: continuous
- 4. horsepower: continuous
- 5. weight: continuous
- 6. acceleration: continuous
- 7. model year: multi-valued discrete
- 8. origin: multi-valued discrete
- 9. car name: string (unique for each instance)

← → ↻ archive.ics.uci.edu/ml/datasets/auto+mpg


Color Scriptor Google Solutions to Introd... 02. Python Strings... DeepLearningZero... TigerCowDoor - 연... PinkWink - M...



Machine Learning Repository
Center for Machine Learning and Intelligent Systems

Auto MPG Data Set
Download: [Data Folder](#) [Data Set Description](#)

Abstract: Revised from CMU StatLib library, data concerns city-cycle fuel consumption



Data Set Characteristics:	Multivariate	Number of Instances:	398	Area:	N/A
Attribute Characteristics:	Categorical, Real	Number of Attributes:	8	Date Donated	1993-07-07
Associated Tasks:	Regression	Missing Values?	Yes	Number of Web Hits:	502080

<https://archive.ics.uci.edu/ml/datasets/auto+mpg>

데이터 정제

```
#%%  
# 데이터 정제 하기  
# 비어있는 칼럼의 행의 수 알아내기  
cnt = dataset.isna().sum()  
print(cnt)  
  
#%%  
# 비어있는 행 제거  
dataset = dataset.dropna()  
  
# 열 'origin'을 빼내 origin에 저장  
origin = dataset.pop('origin')  
print(origin)  
  
#%%  
# "origin" 열은 수치형이 아니고 범주형이므로 원-핫 인코딩(one-hot encoding)으로  
# 변환  
dataset['USA'] = (origin == 1)*1.0  
dataset['Europe'] = (origin == 2)*1.0  
dataset['Japan'] = (origin == 3)*1.0  
dataset.tail(10)
```

데이터셋을 훈련 세트와 테스트 세트로 분할

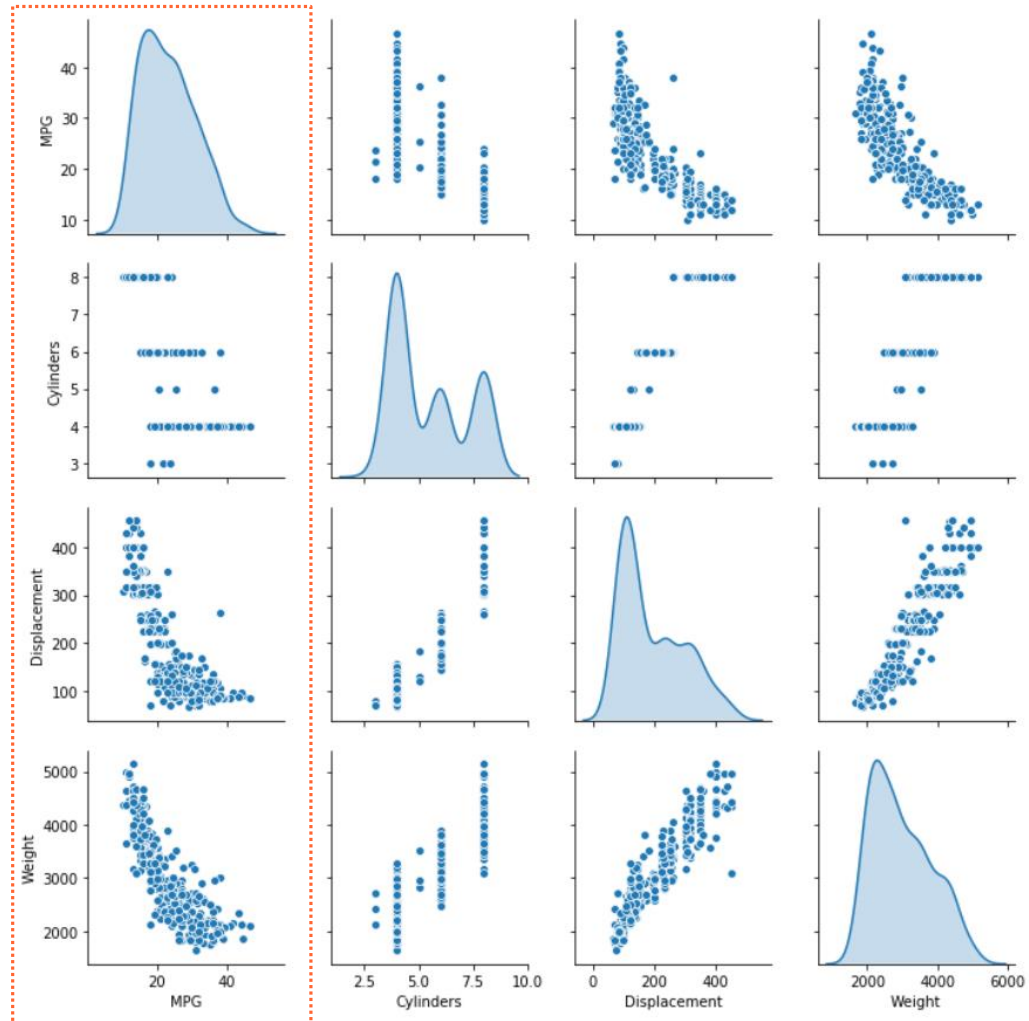
- 80:20으로

- 테스트 세트는 모델을 최종적으로 평가할 때 사용

```
# 데이터셋을 훈련 세트와 테스트 세트로 분할  
# 전체 자료에서 80%를 훈련 데이터로 사용  
train_dataset = dataset.sample(frac=0.8, random_state=0)  
print(train_dataset)  
# 전체 자료에서 나머지 20%를 테스트 데이터로 사용  
test_dataset = dataset.drop(train_dataset.index)  
print(test_dataset)
```

시본의 산점도

- 대각선 `diag_kind`
 - 커널밀도추정곡선
 - kde
 - Kernel Density Estimation
- 색상
 - `palette='bright'`
 - pastel, bright, deep, muted, colorblind, dark



```
##%%
```

```
sns.pairplot(train_dataset[["MPG", "Cylinders", "Displacement", "Weight"]],  
diag_kind="kde")
```

전반적인 통계 확인

- describe()

In [36]:

```
1 # 전반적인 통계도 확인
2 train_stats = train_dataset.describe()
3 print(train_stats)
```

	MPG	Cylinders	Displacement	Horsepower	Weight	#
count	314.000000	314.000000	314.000000	314.000000	314.000000	
mean	23.310510	5.477707	195.318471	104.869427	2990.251592	
std	7.728652	1.699788	104.331589	38.096214	843.898596	
min	10.000000	3.000000	68.000000	46.000000	1649.000000	
25%	17.000000	4.000000	105.500000	76.250000	2256.500000	
50%	22.000000	4.000000	151.000000	94.500000	2822.500000	
75%	28.950000	8.000000	265.750000	128.000000	3608.000000	
max	46.600000	8.000000	455.000000	225.000000	5140.000000	

	Acceleration	Model Year	USA	Europe	Japan
count	314.000000	314.000000	314.000000	314.000000	314.000000
mean	15.559236	75.898089	0.624204	0.178344	0.197452
std	2.789230	3.675642	0.485101	0.383413	0.398712
min	8.000000	70.000000	0.000000	0.000000	0.000000
25%	13.800000	73.000000	0.000000	0.000000	0.000000
50%	15.500000	76.000000	1.000000	0.000000	0.000000
75%	17.200000	79.000000	1.000000	0.000000	0.000000
max	24.800000	82.000000	1.000000	1.000000	1.000000

In [37]:

```
1 train_stats.pop("MPG")
2 train_stats = train_stats.transpose()
3 train_stats.head(9)
```

Out [37]:

	count	mean	std	min	25%	50%	75%	max
Cylinders	314.0	5.477707	1.699788	3.0	4.00	4.0	8.00	8.0
Displacement	314.0	195.318471	104.331589	68.0	105.50	151.0	265.75	455.0
Horsepower	314.0	104.869427	38.096214	46.0	76.25	94.5	128.00	225.0
Weight	314.0	2990.251592	843.898596	1649.0	2256.50	2822.5	3608.00	5140.0
Acceleration	314.0	15.559236	2.789230	8.0	13.80	15.5	17.20	24.8
Model Year	314.0	75.898089	3.675642	70.0	73.00	76.0	79.00	82.0
USA	314.0	0.624204	0.485101	0.0	0.00	1.0	1.00	1.0
Europe	314.0	0.178344	0.383413	0.0	0.00	0.0	0.00	1.0
Japan	314.0	0.197452	0.398712	0.0	0.00	0.0	0.00	1.0

데이터 정규화

- 특성의 스케일과 범위가 다르면 정규화(normalization)하는 것이 권장
 - 의도적으로 훈련 세트만 사용하여 통계치를 생성
 - 테스트 세트를 정규화할 때에도 사용
 - 테스트 세트를 모델이 훈련에 사용했던 것과 동일한 분포로 투영하기 위해서
- 정규화된 데이터를 사용하여 모델을 훈련
 - 입력 데이터를 정규화하기 위해 사용한 통계치(평균과 표준편차)는 원-핫 인코딩과 마찬가지로 모델에 주입되는 모든 데이터에 적용

```
In [55]: 1 train_labels = train_dataset.pop('MPG')
          2 test_labels = test_dataset.pop('MPG')
          3
          4 def norm(x):
          5     return (x - train_stats['mean']) / train_stats['std']
          6
          7 normed_train_data = norm(train_dataset)
          8 normed_test_data = norm(test_dataset)
          9 normed_train_data.tail(10)
```

Out [55]:

	Cylinders	Displacement	Horsepower	Weight	Acceleration	Model Year	USA	Europe	Japan
171	-0.869348	-0.587727	-0.232816	-0.341571	-0.738281	-0.244335	-1.286751	-0.465148	2.012852
70	1.483887	1.961837	2.234620	1.696588	-1.096803	-1.060519	0.774676	-0.465148	-0.495225
307	0.307270	-0.213919	0.265921	-0.343941	-0.953394	0.843910	0.774676	-0.465148	-0.495225
49	-0.869348	-0.702745	-0.495310	-0.912730	-0.559020	-1.332580	0.774676	-0.465148	-0.495225
209	-0.869348	-0.721914	-0.442811	0.331495	2.273303	0.027726	-1.286751	2.143005	-0.495225
281	0.307270	0.044872	-0.521559	-0.000298	0.946772	0.843910	0.774676	-0.465148	-0.495225
229	1.483887	1.961837	1.972127	1.457223	-1.598734	0.299787	0.774676	-0.465148	-0.495225
150	-0.869348	-0.836932	-0.311564	-0.710099	-0.021237	-0.516397	-1.286751	-0.465148	2.012852
145	-0.869348	-1.076553	-1.151543	-1.169870	1.233589	-0.516397	-1.286751	-0.465148	2.012852
182	-0.869348	-0.846517	-0.495310	-0.623596	-0.021237	0.027726	-1.286751	2.143005	-0.495225

모델을 구성

- 두 개의 완전 연결(densely connected) 은닉층으로 Sequential 모델
 - 출력 층은 하나의 연속적인 값을 반환
 - 나중에 두 번째 모델을 만들기 쉽도록 build_model 함수로 모델 구성 단계를 감쌌

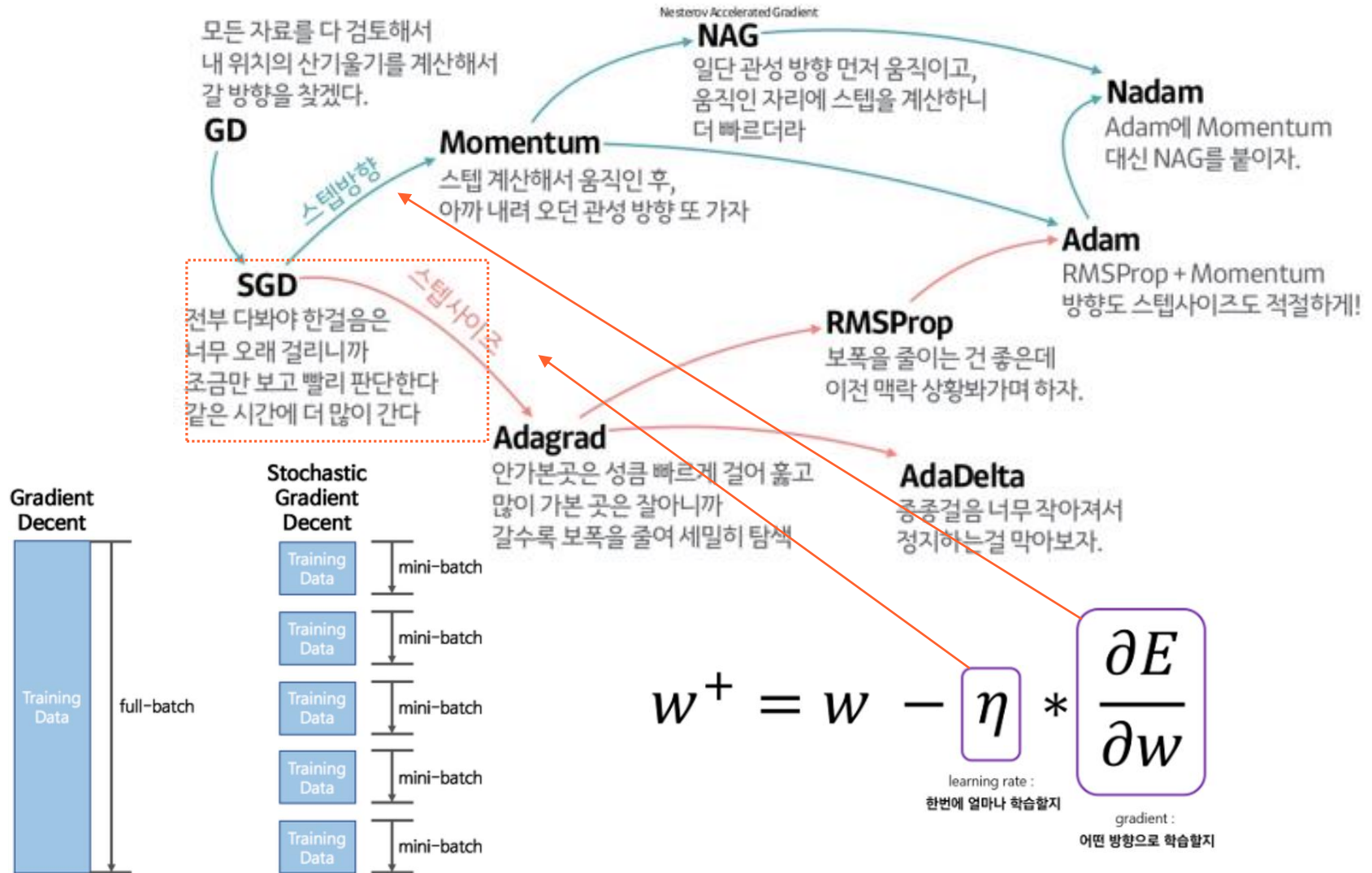
```
In [56]: 1 def build_model():
2         model = keras.Sequential([
3             layers.Dense(64, activation='relu', input_shape=[len(train_dataset.keys())]),
4             layers.Dense(64, activation='relu'),
5             layers.Dense(1)
6         ])
7         optimizer = tf.keras.optimizers.RMSprop(0.001)
8         model.compile(loss='mse', optimizer=optimizer, metrics=['mae', 'mse'])
9         return model
10
11 model = build_model()
12 model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 64) $(9 + 1) * 64$	640
dense_1 (Dense)	(None, 64) $(64 + 1) * 64$	4160
dense_2 (Dense)	(None, 1) $(64 + 1) * 1$	65

Total params: 4,865
 Trainable params: 4,865
 Non-trainable params: 0

옵티마이저 발전 과정



모델을 한번 실행

- 훈련 세트에서 10 샘플을 하나의 배치로 만들어
 - model.predict 메서드를 호출

```
In [59]: 1 example_batch = normed_train_data[:10]
          2 example_result = model.predict(example_batch)
          3 print(example_result)
```

WARNING:tensorflow:Falling back from v2 loop because of error: Failed to find data adapter that can handle input: <class 'pandas.core.frame.DataFrame'>, <class 'NoneType'>

```
[[-0.10031918]
 [-0.2947113 ]
 [-0.5550761 ]
 [-0.66249985]
 [-0.2611794 ]
 [ 0.08221942]
 [-0.27615458]
 [-0.2320385 ]
 [ 0.12494856]
 [ 0.23880696]]
```


콜백

- 학습 과정의 한 에폭마다 적용할 함수의 세트
 - 학습의 각 단계에서 콜백의 적절한 메서드가 호출
 - 모델의 내적 상태와 통계자료를 확인
 - 키워드 인수 callbacks
 - Sequential이나 Model 클래스의 .fit() 메서드에 전달이 가능

에폭크가 끝날 때마다 점(.)을 출력, 100번마다 다음 줄로 이동해 훈련 진행 과정을 표시합니다

```
class PrintDot(keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs):
        if epoch % 100 == 0: print('')
        print('.', end='')
```

```
EPOCHS = 1000
```

```
history = model.fit(
    normed_train_data, train_labels,
    epochs=EPOCHS, validation_split = 0.2, verbose=0,
    callbacks=[PrintDot()])
```

history 객체에 저장된 통계치를 사용

```
In [62]: 1 hist = pd.DataFrame(history.history)
          2 hist['epoch'] = history.epoch
          3 hist.tail(10)
```

Out [62] :

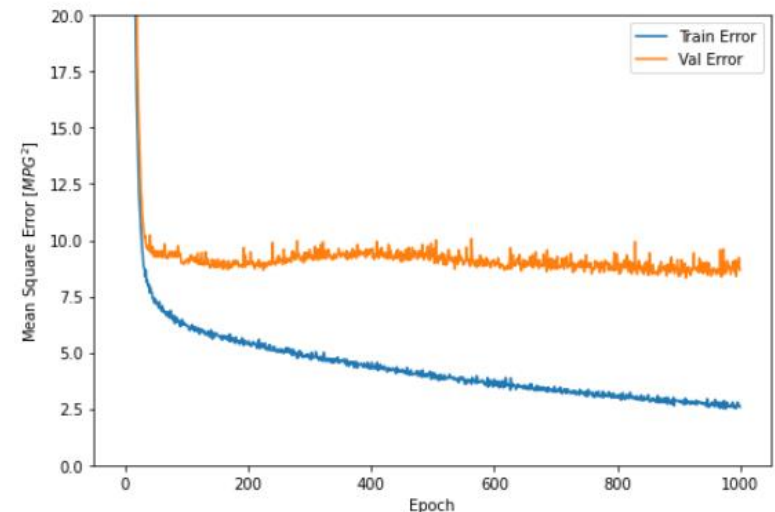
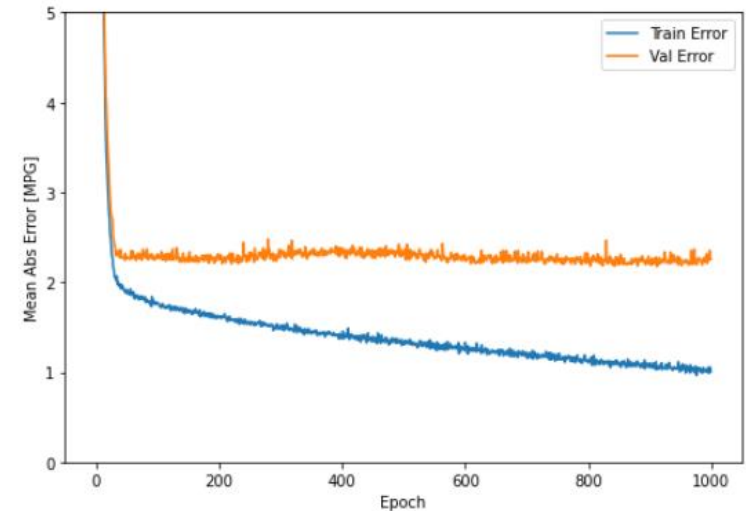
	훈련 데이터			검증 데이터			
	loss	mae	mse	val_loss	val_mae	val_mse	epoch
990	0.900445	0.586849	0.900445	11.896770	2.503707	11.896770	990
991	1.009565	0.595112	1.009565	11.819691	2.453713	11.819692	991
992	0.967227	0.620926	0.967227	12.608477	2.664422	12.608477	992
993	1.088160	0.654080	1.088160	12.058443	2.535334	12.058444	993
994	1.110952	0.659188	1.110952	11.479473	2.406013	11.479473	994
995	1.005472	0.609855	1.005472	11.587123	2.428143	11.587123	995
996	0.833751	0.529896	0.833751	11.629765	2.428819	11.629766	996
997	0.928984	0.586742	0.928984	11.978379	2.466970	11.978379	997
998	0.952604	0.614702	0.952603	11.525044	2.422141	11.525043	998
999	0.980738	0.642155	0.980738	12.360261	2.618787	12.360261	999

훈련 과정을 시각화

검증 손실이 계속 감소하는 것이 중요

In [63]:

```
1 def plot_history(history):
2     hist = pd.DataFrame(history.history)
3     hist['epoch'] = history.epoch
4     plt.figure(figsize=(8,12))
5
6     plt.subplot(2,1,1)
7     plt.xlabel('Epoch')
8     plt.ylabel('Mean Abs Error [MPG]')
9     plt.plot(hist['epoch'], hist['mae'], label='Train Error')
10    plt.plot(hist['epoch'], hist['val_mae'], label='Val Error')
11    plt.ylim([0,5])
12    plt.legend()
13
14    plt.subplot(2,1,2)
15    plt.xlabel('Epoch')
16    plt.ylabel('Mean Square Error [MPG^2]')
17    plt.plot(hist['epoch'], hist['mse'], label='Train Error')
18    plt.plot(hist['epoch'], hist['val_mse'], label='Val Error')
19    plt.ylim([0,20])
20    plt.legend()
21    plt.show()
22
23 plot_history(history)
24
```



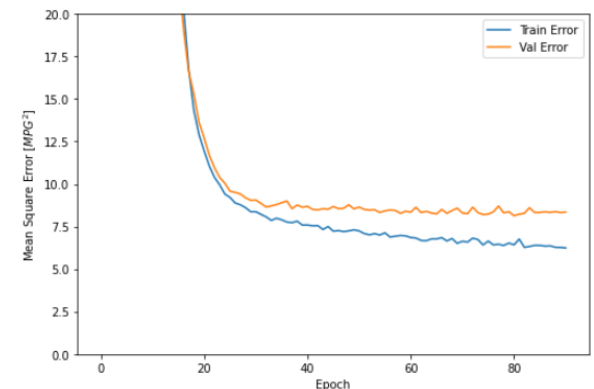
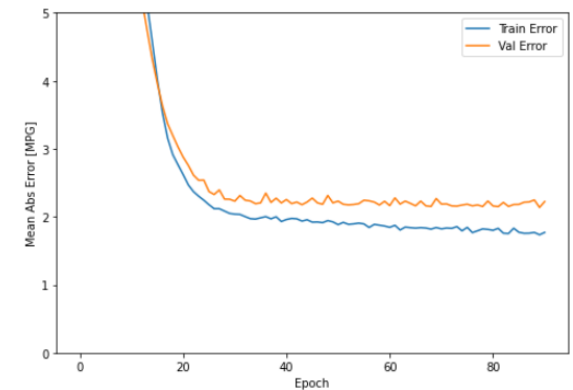
EarlyStopping 콜백

• EarlyStopping 콜백(callback)

- model.fit 메서드를 수정하여 검증 점수가 향상되지 않으면 자동으로 훈련을 멈추도록
 - 지정된 에POCH 횟수 동안 성능 향상이 없으면 자동으로 훈련이 멈추도록
 - 옵션 **monitor, patience**
 - 손실 **val_loss**가 10회 초과해 감소하지 않으면 중단

```
In [65]: 1 model = build_model()
2
3 # patience 매개변수는 성능 향상을 체크할 에POCH 횟수입니다
4 early_stop = keras.callbacks.EarlyStopping(monitor='val_loss', patience=10)
5 history = model.fit(normed_train_data, train_labels, epochs=EPOCHS,
6                     validation_split = 0.2, verbose=0, callbacks=[early_stop, PrintDot()])
7
8 plot_history(history)
```

☞



모델의 성능을 확인

- 테스트 세트에서 모델의 성능을 확인

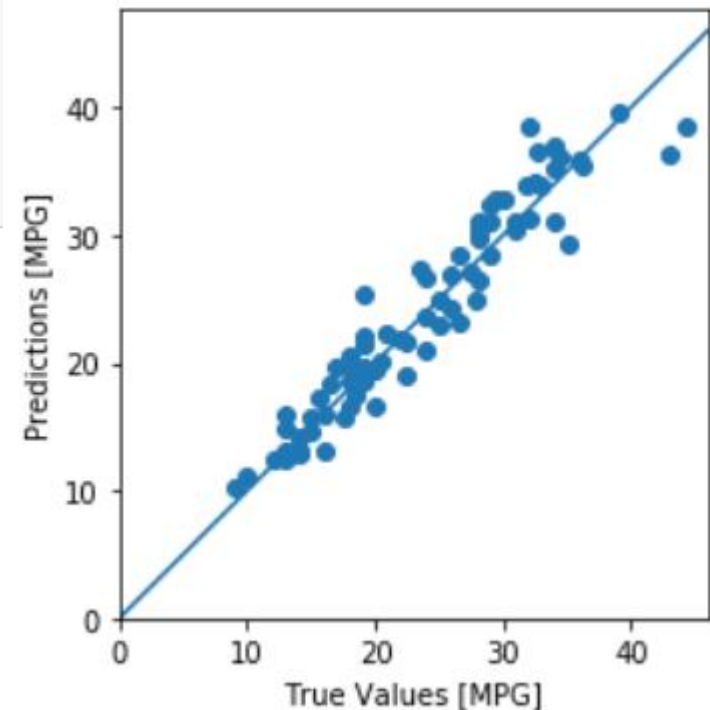
```
In [67]: 1 loss, mae, mse = model.evaluate(normed_test_data, test_labels, verbose=2)
          2 print("테스트 세트의 평균 절대 오차: {:.2f} MPG".format(mae))
```

```
WARNING:tensorflow:Falling back from v2 loop because of error: Failed to find data adapter that can handle input: <class 'pandas.core.frame.DataFrame'>, <class 'NoneType'>
78/78 - 0s - loss: 6.0765 - mae: 1.9148 - mse: 6.0765
테스트 세트의 평균 절대 오차: 1.91 MPG
```

예측(1)

- 테스트 세트에 있는 샘플을 사용해 MPG 값을 예측

```
In [68]: 1 # 테스트 세트에 있는 샘플을 사용해 MPG 값을 예측
          2 test_predictions = model.predict(normed_test_data).flatten()
          3
          4 plt.scatter(test_labels, test_predictions)
          5 plt.xlabel('True Values [MPG]')
          6 plt.ylabel('Predictions [MPG]')
          7 plt.axis('equal')
          8 plt.axis('square')
          9 plt.xlim([0, plt.xlim()[1]])
         10 plt.ylim([0, plt.ylim()[1]])
         11 _ = plt.plot([-100, 100], [-100, 100])
```



메소드 axis()

axis(*v, **kwargs)

axis() 함수는 x,y축의 범위를 설정할 수 있게 하는 것과 동시에 여러 옵션을 설정할 수 있는 함수이다.

*v자리에는 리스트나 튜플같이 순서가 있는 시퀀스형 자료형이 입력된다.

함수 입력 변수)

```
axis([xmin, xmax, ymin, ymax]) # x,y 축의 범위 설정
axis(option)                  # 축의 옵션(속성)을 설정
axis(emit)                    # 축의 범위 변경의 인지여부를 설정
```

입력방식은 축의 범위를 설정할 경우에는 리스트나 튜플로 입력하여야 한다.

emit은 축의 범위를 변화시킬 시 이를 인지하여 그래프에 반영할지 여부를 결정해주는 변수로 기본값은 True이다.

option은 키워드 인자로 입력하여야 하며 문자열(string,str)을 입력으로 받고, 다음표와 같은 옵션들이 입력 가능하다.

값	설명
'on'	축과 라벨을 켜다.
'off'	축과 라벨을 끈다.
'equal'	각 축의 범위와 축의 스케일을 동일하게 설정한다.
'scaled'	플롯 박스의 차원과 동일하게 축의 스케일을 설정한다.
'tight'	모든 데이터를 볼 수 있을 정도로 축의 범위를 충분히 크게 설정한다.
'auto'	축의 스케일을 자동으로 설정한다.
'normal'	'auto'와 동일하다.
'image'	데이터 범위에 대해 축의 범위를 사용한 'scaled'이다.
'square'	각축의 범위 즉 $x_{max}-x_{min}=y_{max}-y_{min}$ 되도록 설정한다.

예측(2)

- 오차의 분포

```
In [69]: 1 error = test_predictions - test_labels  
2 plt.hist(error, bins = 25)  
3 plt.xlabel("Prediction Error [MPG]")  
4 _ = plt.ylabel("Count")
```

