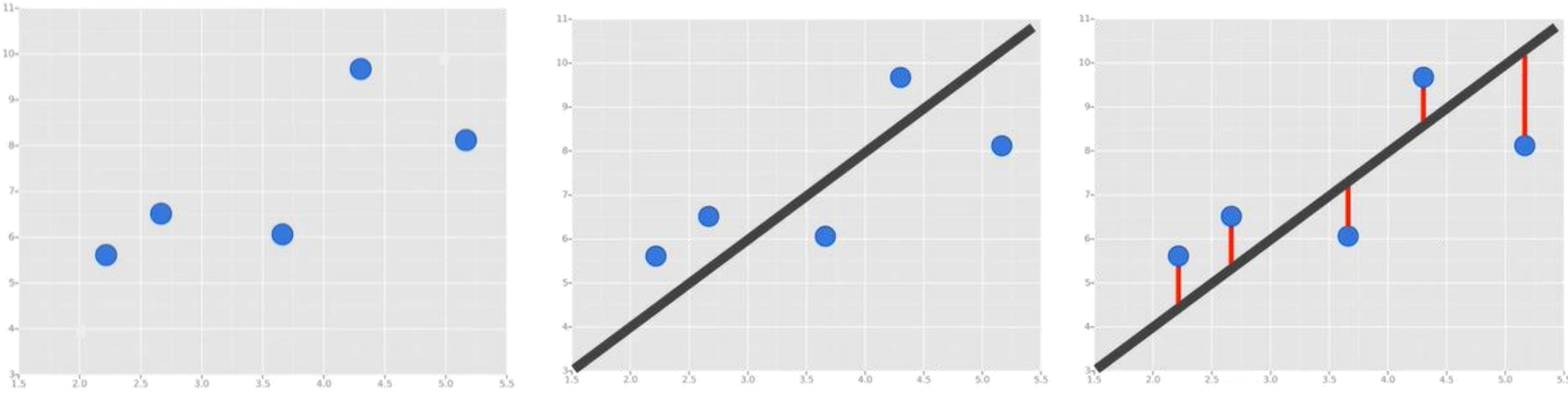
# Neural Network Regression Model


박경규

# 선형 회귀

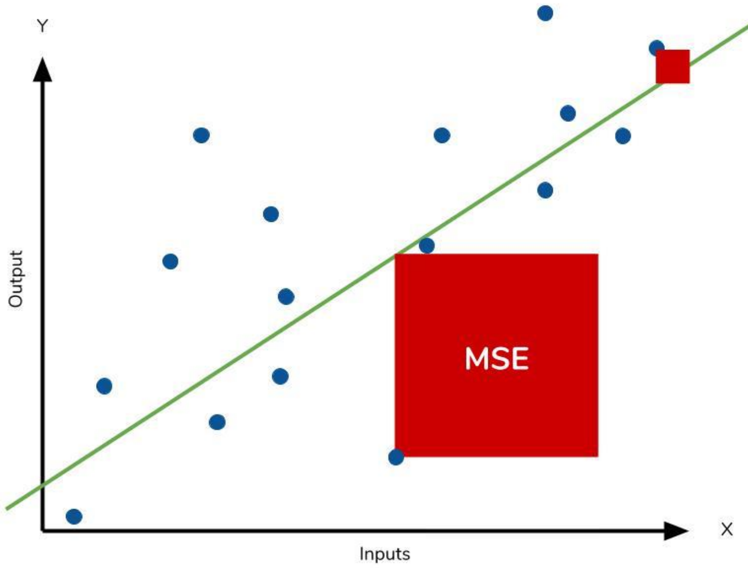
**종속 변수 y와 한 개 이상의 독립 변수 X와의 선형 상관 관계를 모델링 하는 회귀분석 기법**



# 손실함수(Loss Function)

**회귀모델(Regression)에서는 주로 평균제곱오차(MSE)를 손실함수로 사용합니다.**

## ■ MSE(Mean Squared Error)

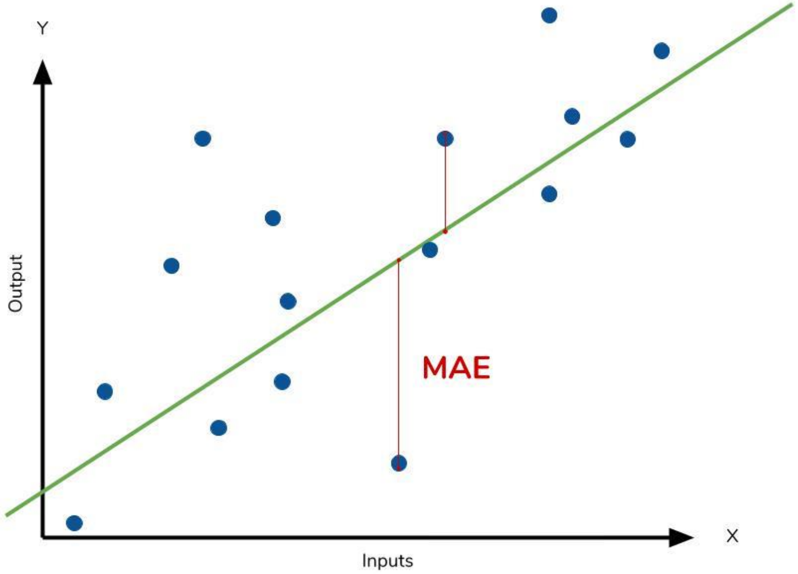



$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

레이블 값
(실제값)

y_hat
(모델이 예측한 값)

## ■ MAE(Mean Absolute Error)




$$\frac{1}{n}\sum_{i=1}^{n}(|y_i - \hat{y}_i|)$$

참고자료 : https://mizykk.tistory.com/102

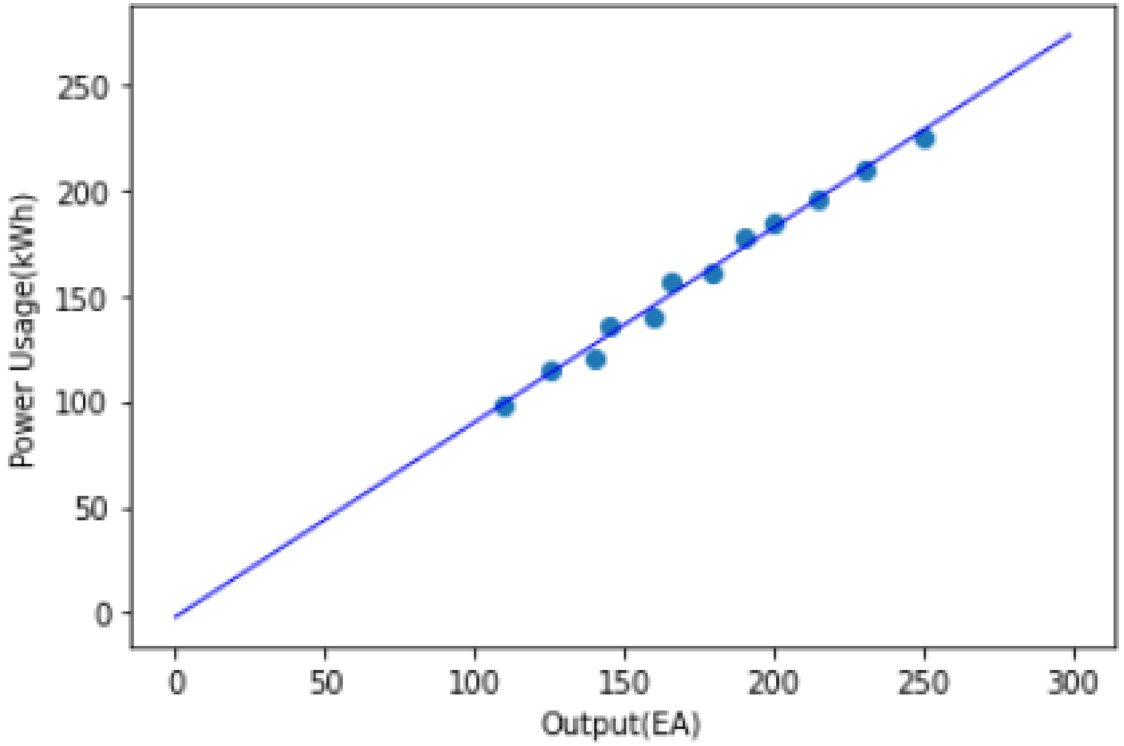
# 선형 회귀 - scipy.stats

```python
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt

# 생산량
output = [110, 125, 140, 145, 160, 166, 179, 190, 200, 215, 230, 250]

# 전력사용량
power_usage = [98, 115, 120, 136, 140, 156, 160, 177, 185, 195, 210, 225]

# p-value   : 유의 확률, 일반적으로 0.05 미만일 때 유의미
slope, intercept, r_value, p_value, stderr = stats.linregress(output, power_usage)
```

# 선형 회귀 – scipy.stats

```python
# 생산량 134개일 때 전기사용량 예측
product = 134
print("기울기(slope) : ", slope)
print("절편(intercept) : ", intercept)
print("상관계수(r_value) : ", r_value)
print("유의확률(p_value) : ", p_value )
print("{}개 => 예측량 {}kWh".format(
      product, product*slope + intercept))

plt.scatter(output, power_usage)
x = np.arange(0, 300)
y = [(slope*num + intercept) for num in x]
plt.plot(x, y, 'b', lw=1)
plt.xlabel("Output(EA)")
plt.ylabel("Power Usage(kWh)")
plt.show()
```

```
기울기(slope) :  0.9200457304535211
절편(intercept) :  -2.024707604744151
상관계수(r_value) :  0.995041535282884
유의확률(p_value) :  2.340961379756715 5e-11
134개 => 예측량 121.26142027602768kWh
```



# 선형 회귀 – sklearn.linear_model

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
X = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ]).reshape(-1,1)
y = np.array([13, 25, 34, 47, 59, 62, 79, 88, 90, 100])
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                        test_size=0.3, random_state=42)
model = LinearRegression()
model.fit(X_train, y_train)
predictions = model.predict(X_test)
```

# 선형 회귀 - sklearn.linear_model

regression.ipynb

```
[1]  import numpy as np
```

```
[2]  # create dummy data for training
     x_values = [i for i in range(11)]
     print(x_values)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
[3]  # Convert to numpy
     x_train = np.array(x_values, dtype=np.float32)
     print(x_train.shape)
```

```
(11,)
```

```
[4]  # IMPORTANT: 2D required
     x_train = x_train.reshape(-1, 1)
     print(x_train.shape)
```

```
(11, 1)
```

# 선형 회귀 – 신경망

```
[5]  y_values = [2*i + 1 for i in x_values]
```

```
[6]  print(y_values)
```

```
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21]
```

```
[7]  y_train = np.array(y_values, dtype=np.float32)
     print(y_train.shape)
```

```
(11,)
```

```
[8]  # IMPORTANT: 2D required
     y_train = y_train.reshape(-1, 1)
     print(y_train.shape)
```

```
(11, 1)
```

# 선형 회귀 – 신경망

## ■ 신경망 모델 아키텍처

```
[9]  import torch
     from torch.autograd import Variable
```

```
[10] class linearRegression(torch.nn.Module):
         def __init__(self, input_size, output_size):
             super(linearRegression, self).__init__()
             self.linear = torch.nn.Linear(input_size, output_size)

         def forward(self, x):
             out = self.linear(x)
             return out
```

# 선형 회귀 – 신경망

```
[11] input_dim = 1
     output_dim = 1
     learning_rate = 0.01
     epochs = 100

     model = linearRegression(input_dim, output_dim)
     # For GPU
     if torch.cuda.is_available():
         model.cuda()
```

```
[12] criterion = torch.nn.MSELoss()
     optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
```

# 선형 회귀 – 신경망

## ■ 신경망 모델 훈련

```
[13] for epoch in range(epochs):
         # Converting inputs and labels to Variable
         if torch.cuda.is_available():
             inputs = Variable(torch.from_numpy(x_train).cuda())
             labels = Variable(torch.from_numpy(y_train).cuda())
         else:
             inputs = Variable(torch.from_numpy(x_train))
             labels = Variable(torch.from_numpy(y_train))

         # Clear gradient buffers because we don't want any gradient from previous epoch
         optimizer.zero_grad()

         # get output from the model, given the inputs
         outputs = model(inputs)
```

# 선형 회귀 – 신경망

## ■ 신경망 모델 훈련

[13]

```
    # get loss for the predicted output
    loss = criterion(outputs, labels)
    print(loss)
    # get gradients w.r.t to parameters
    loss.backward()

    # update parameters
    optimizer.step()

    print('epoch {}, loss {}'.format(epoch, loss.item()))
```

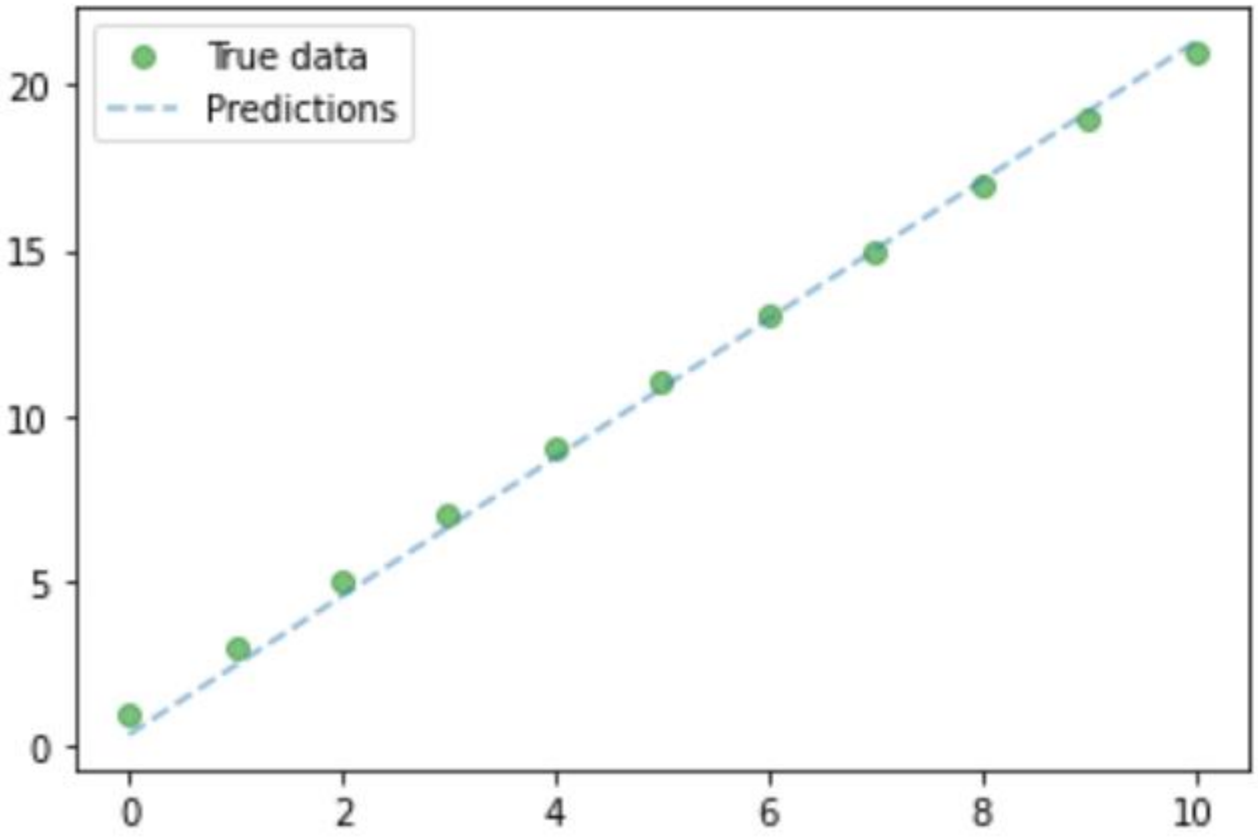
```
tensor(135.7494, grad_fn=<MseLossBackward>)
epoch 0, loss 135.74935913085938
tensor(11.4099, grad_fn=<MseLossBackward>)
epoch 1, loss 11.40990924835205
tensor(1.2642, grad_fn=<MseLossBackward>)
```

# 선형 회귀 – 신경망

## ■ 신경망 모델 사용(예측)

```
[14] with torch.no_grad(): # we don't need gradients in the testing phase
         if torch.cuda.is_available():
             predicted = model(Variable(torch.from_numpy(x_train).cuda())).cpu().data.numpy()
         else:
             predicted = model(Variable(torch.from_numpy(x_train))).data.numpy()
         print(predicted)

     import matplotlib.pyplot as plt

     plt.clf()
     plt.plot(x_train, y_train, 'go', label='True data', alpha=0.5)
     plt.plot(x_train, predicted, '--', label='Predictions', alpha=0.5)
     plt.legend(loc='best')
     plt.show()
```

# 선형 회귀 – 신경망



# Thank you

kgpark88@gmail.com