**교육일지**

**교육 제목 : 딥러닝**

**교육 장소 : YGL C6 강의실**

**교육 일시 : 2021/10/26**

**Normalization**

**data = pd.read\_csv('./test.csv')**

**a = data['D']**

**x\_data = data.copy()**

**y\_data = x\_data.pop("D")**

**# 데이터 분포도 확인하기**

**sns.pairplot(data[["A","B","C","D"]], diag\_kind='hist')**

**plt.show()**

**# data의 min, max, mean, std값 구하기**

**dataset\_stats = data.describe()**

**dataset\_stats = dataset\_stats.transpose()**

**#data.min()**

**#data.max()**

**#data.mean()**

**## data normalization**

**def min\_max\_norm(x):**

**return (x - dataset\_stats['min']) / (dataset\_stats['max'] - dataset\_stats['min'])**

**def standard\_norm(x):**

**return (x - dataset\_stats['mean']) / dataset\_stats['std']**

**min\_max\_norm\_train\_data = min\_max\_norm(data)**

**standard\_norm\_train\_data = standard\_norm(data)**

**print("min max : ")**

**print(min\_max\_norm\_train\_data)**

**print("standard :")**

**print(standard\_norm\_train\_data)**

**BostonHousing Normalization**

**data = pd.read\_csv('./BostonHousing.csv')**

**x\_data = data.copy()**

**y\_data = data.pop('MEDV')**

**dataset\_stats = data.describe()**

**dataset\_stats = dataset\_stats.transpose()**

**## data normalization**

**def min\_max\_norm(x):**

**return (x - dataset\_stats['min']) / (dataset\_stats['max'] - dataset\_stats['min'])**

**def standard\_norm(x):**

**return (x - dataset\_stats['mean']) / dataset\_stats['std']**

**normed\_train\_data = standard\_norm(data)**

**input\_layer = tf.keras.layers.Input(shape=(13,))**

**x = tf.keras.layers.Dense(50, activation='sigmoid')(input\_layer)**

**x = tf.keras.layers.Dense(100, activation='sigmoid')(x)**

**x = tf.keras.layers.Dense(300, activation='sigmoid')(x)**

**output\_layer = tf.keras.layers.Dense(1, activation=None)(x)**

**model = tf.keras.Model(inputs=[input\_layer], outputs=[output\_layer])**

**model.summary()**

**loss = tf.keras.losses.mean\_squared\_error**

**optimizer = tf.keras.optimizers.SGD(learning\_rate=0.007)**

**metrics = tf.keras.metrics.RootMeanSquaredError()**

**model.compile(loss=loss,**

**optimizer=optimizer,**

**metrics=[metrics])**

**result = model.fit(normed\_train\_data, y\_data, epochs = 1000, batch\_size=100)**

**print(result.history.keys())**

**dict\_keys(['loss', 'root\_mean\_squared\_error'])**

**loss = result.history['loss']**

**# loss 그래프**

**epochs = range(1, len(loss)+1)**

**plt.subplot(211)**

**plt.plot(epochs, loss, 'b-', label='Training loss')**

**plt.title('Training loss')**

**plt.xlabel('Epochs')**

**plt.ylabel('Loss')**

**plt.legend()**

**mae = result.history['root\_mean\_squared\_error']**

**epochs = range(1, len(mae)+1)**

**# mae그래프**

**plt.subplot(212)**

**plt.plot(epochs, mae, 'r-', label='Training mae')**

**plt.title('Training rmse')**

**plt.xlabel('Epochs')**

**plt.ylabel('rmse')**

**plt.legend()**

**print("\ Test rmse : %.4f" % (model.evaluate(normed\_train\_data, y\_data)[1]))**

**BostonHousing split**

**x\_data = data.copy()**

**ori\_y = x\_data.pop('MEDV')**

**from sklearn. import train\_test\_split**

**x\_train1, x\_test, y\_train1, y\_test = train\_test\_split(xmodel\_selection\_data, ori\_y, test\_size=0.3, shuffle=True)**

**x\_train, x\_valid, y\_train, y\_valid = train\_test\_split(x\_train1, y\_train1, test\_size=0.2, shuffle=True)**

**input\_layer = tf.keras.layers.Input(shape=(13,))**

**x = tf.keras.layers.Dense(50, activation='sigmoid')(input\_layer)**

**x = tf.keras.layers.Dense(100, activation='sigmoid')(x)**

**x = tf.keras.layers.Dense(300, activation='sigmoid')(x)**

**out\_layer = tf.keras.layers.Dense(1, activation=None)(x)**

**model = tf.keras.Model(inputs=[input\_layer], outputs=[out\_layer])**

**model.summary()**

**input\_layer = tf.keras.layers.Input(shape=(13,))**

**x = tf.keras.layers.Dense(50, activation='sigmoid')(input\_layer)**

**x = tf.keras.layers.Dense(100, activation='sigmoid')(x)**

**x = tf.keras.layers.Dense(300, activation='sigmoid')(x)**

**out\_layer = tf.keras.layers.Dense(1, activation=None)(x)**

**model = tf.keras.Model(inputs=[input\_layer], outputs=[out\_layer])**

**model.summary()**

**result = model.fit(x\_train, y\_train, epochs = 200, batch\_size = 10, validation\_data=(x\_valid, y\_valid))**

**print(result.history.keys())**

**loss = result.history['loss']**

**val\_loss = result.history['val\_loss']**

**### loss와 val\_loss를 그래프화**

**epochs = range(1, len(loss) + 1)**

**plt.subplot(211) ## 2x1 개의 그래프 중에 1번째**

**plt.plot(epochs, loss, 'b-', label='Training loss')**

**plt.plot(epochs, val\_loss, 'r', label='Validation loss')**

**plt.title('Training and validation loss')**

**plt.xlabel('Epochs')**

**plt.ylabel('Loss')**

**plt.legend()**

**### history에서 mean\_absolute\_error val\_mean\_absolute\_error key를 가지는 값들만 추출**

**rmse = result.history['root\_mean\_squared\_error']**

**val\_rmse = result.history['val\_root\_mean\_squared\_error']**

**epochs = range(1, len(rmse) + 1)**

**### mean\_absolute\_error val\_mean\_absolute\_error key를 그래프화**

**plt.subplot(212) ## 2x1 개의 그래프 중에 2번째**

**plt.plot(epochs, rmse, 'b-', label='Training rmse')**

**plt.plot(epochs, val\_rmse, 'r', label='Validation rmse')**

**plt.title('Training and validation rmse')**

**plt.xlabel('Epochs')**

**plt.ylabel('rmse')**

**plt.legend()**

**print("\n Test rmse: %.4f" % (model.evaluate(x\_test, y\_test)[1]))**

**plt.show()**