머신러닝실습 레포트

-6주차-

제출일 : 2022.06.05.

학 번 : 2017142012

이 름 : 김효건

# **0. 목차**

1. **목차**
2. **실습 #1**
   1. Error Back-Propagation 알고리즘 구현
3. **실습 #2**
   1. 기초 데이터 정제
   2. Train set 학습 및 결과
   3. 결과 보는 함수
4. **실습 #3**
   1. Test set 결과

# **1. 실습 #1**

**1.1 Error Back-Propagation 알고리즘 구현.**

**[Layer층 클래스]**

|  |
| --- |
| # 각 층을 클래스로 설정  class MultiPerceptron:      def \_\_init\_\_(self, last\_layer=None, node=0, activation\_param="none", name=None):          """\_summary\_          Args:              last\_layer: 이전 층, 이전층의 데이터 값과 가중치를 통해 입력값을 계산하기 위해 가져옴              node: 사용자 설정 값, 원하는 노드 수 설정              activation\_param: 활성화 함수 string 변수              name: 각 층의 이름 설정          """          self.tolerance = 1e-15    # 허용 오차            # 각 활성화 함수 string변수와 함수 매핑          self.activate\_func = {              "none": self.none,              "relu": self.relu,              "sigmoid": self.sigmoid,              "step": self.step,              "tanh": self.tanh,              "identity": self.identity,              "softmax": self.softmax          }            self.last\_layer = last\_layer # 이전 층          # 현재 층의 활성 함수 종류 저장          self.activation\_param = activation\_param          # 활성화 함수 매핑          self.activate = self.activate\_func[activation\_param]          self.hidden\_node = node # 은닉층의 노드의 갯수          self.in\_data = None     # 이전 층의 데이터와 가중치의 행렬 계산한 값이 들어온다.          self.w = None           # 현재 층의 가중치          self.out\_data = None    # 입력으로 들어온 데이터를 활성화 함수를 거치게 한 값 다음 층이나 결과가 된다.            self.name=name # Debug용, 각 은닉층의 이름 지정        # Forward propagation      def neuron(self):          """\_summary\_              퍼셉트론의 계산을 진행하는 주요 함수              이전층의 출력값과 가중치 값의 행렬 계산 => 입력 => 활성화 함수 => 출력 => (다음 층)          """          # 이전 레이어와 가중치의 곱으로 퍼셉트론 계산          tmp = self.layer\_calc()            # 활성화 전 곱으로 들어온 값, 입력 데이터에 저장          self.in\_data = tmp            # 입력 데이터 활성화해서 넘겨줌          tmp = self.activate(self.in\_data)            # 활성화를 거친 값을 출력 값으로 지정          self.out\_data = tmp          def layer\_calc(self):          """\_summary\_          이전 층의 출력값과 가중치 값의 행렬 계산          Returns: 행렬 계산 값          """            x\_b = np.append(self.last\_layer.out\_data,1)          tmp = x\_b.dot(self.last\_layer.w)            return tmp        # Input later      def none(self, x):          return x        # 아래는 활성화 함수      # 시그모이드 함수      def sigmoid(self, x):          x = 1 / ( 1 + np.exp(-x))          return x        # 시그모이드 미분 값      def sigmoid\_prime(self, x):          return x \* (1-x)      # 계단 함수      def step(self, x):          return x > 0      # 하이퍼볼릭 탄젠트 함수      def tanh(self, x):          return (np.exp(x) - np.exp(-x))\                  / (np.exp(x) + np.exp(-x))        # 탄젠트 미분 함수      def tanh\_prime(self, x):          return 1 - self.tanh(x) \*\* 2      # 항등 함수      def identity(self, x):          return x        # Softmax 함수      def softmax(self, x):          e = np.exp(x - np.max(x))          s = np.sum(e)          return e / s        # ReLU함수      def relu(self, x):          return np.array(list(map(lambda d: d if d>0 else 0, x)))        def relu\_prime(self, x):          return np.array(list(map(lambda d: 1 if d>0 else 0, x))) |

**[모델 초기 값]**

|  |
| --- |
| # 모델  class customModel:      """\_summary\_          모든 퍼셉트론 층을 담아두는 함수      """      def \_\_init\_\_(self,x=[], y=[], lr=0.0001, epoch=10000, w=[], test\_x=[], test\_y=[],mean=0, mu=2):          """\_summary          Args:              x: 입력 데이터, 입력데이터는 항상 numpy array로 준다.              y: 레이블 데이터              mean mu, 랜덤값을 위한 평균과 분산              hidden\_layer: 은닉층의 수          """          # 모델 전체 layer와 가중치 담아두는 리스트          self.layer= []          self.weight = w          self.node\_prime\_model = []            # Epoch에 따른 정확도, MSE 리스트          self.accuracy\_list = []          self.mse\_list = []            # 초기화          # 랜덤하게 가중치 생성을 위한 평균과 분산          self.mean = mean          self.mu = mu            # lr, epoch 초기화          self.lr = lr          self.epoch = epoch            self.label=y        # 라벨 값          self.in\_data = x    # 처음 데이터 셋          self.out\_data = []  # 신경망 결과            # 그래프용 & valid 테스트          self.test\_accuracy\_list = []          self.test\_mse\_list = []          self.test\_x = test\_x          self.test\_y = test\_y            self.input\_feature\_cnt = 0 # 입력 데이터의 특성 수          self.output\_class\_cnt = 0  # 출력 결과의 클래스 수          self.hidden\_layer\_cnt = 0  # 은닉층의 수            self.check\_feature\_class\_cnt() # 입력 데이터의 특성 수와 출력 결과 클래스 수 파악        # 모델 클래스 생성시 입력데이터를 통해 입력 특성 수 지정하고      # 라벨데이터를 통해 출력 클래스 개수 정하는 함수        def check\_feature\_class\_cnt(self):          # numpy인 경우에만 사용          if type(self.in\_data) is np.ndarray:              self.input\_feature\_cnt = len(self.in\_data[0])            if type(self.label) is np.ndarray:              self.output\_class\_cnt = len(self.label[0]) |

**[모델 학습 함수 – 메인 동작]**

|  |
| --- |
| # 매 에폭마다 데이터를 셔플하여 데이터 순서를 연관하게 하지 않는다.      def dataShuffle(self):          # 데이터 갯수 만큼 랜덤 인덱스를 만든다.          index = np.random.permutation(len(self.train\_x\_data))          length = len(self.train\_x\_data)          # 각 인덱스에 맞게 라벨과 값을 섞어준다.          temp\_x = self.train\_x\_data.copy()          temp\_y = self.train\_label.copy()          self.train\_x\_data = np.array([temp\_x[index[i]] for i in range(length)] )          self.train\_label = np.array([temp\_y[index[i]] for i in range(length)])        # valid 함수, 나중에 텀프할때 쓸거같아 만들었습니다.      def valid(self):          # 현재 가중치를 통해 valid 데이터를 순전파하여          # 새로운 데이터로 overfitting한지 확인          accuracy = 0          mse = 0          length = len(self.test\_x)          # 한 에폭마다 전체 데이터 셋을 전부 돌아야한다.          for iter in range(len(self.test\_x)):              # 하나의 train데이터가 입력 데이터로 된다.              self.layer[0].in\_data = self.test\_x[iter]              self.layer[0].out\_data = self.test\_x[iter]                # 순전파 시작              self.foward\_propagation()                # 하나의 데이터가 정확한지와 mse를 구한다.              acc, one\_mse, \_, \_ = self.one\_mse\_acc(self.layer[-1], self.test\_y[iter])              accuracy += acc              mse += one\_mse            self.test\_accuracy\_list.append(accuracy / length)          self.test\_mse\_list.append(mse / length)        # 설계한 은닉층과 출력층을 실질적으로 데이터를 넣어 실행시키는 함수      def fit(self):          """              입력 데이터 : row, col          """            # 모델로 들어온 데이터 입력 데이터에 넣는다.          # layer0 = 입력층, 입력층은 아무 것도 설정하지 않으므로 입력 데이터 -> 출력 데이터          length = len(self.in\_data)            # 시작시 train셋에 입력 데이터를 넣느다.          self.train\_x\_data = self.in\_data.copy()          self.train\_label = self.label.copy()          # 가중치 업데이트 반복횟수          for i in range(self.epoch):              # 매 에폭마다 셔플              self.dataShuffle()                # 평균과 정확도 계산을 위한 변수              accuracy = 0              mse = 0                self.valid()              # 한 에폭마다 전체 데이터 셋을 전부 돌아야한다.              for iter in range(len(self.train\_x\_data)):                  # 하나의 train데이터가 입력 데이터로 된다.                  self.layer[0].in\_data = self.train\_x\_data[iter]                  self.layer[0].out\_data = self.train\_x\_data[iter]                    # 순전파 시작                  self.foward\_propagation()                    # 하나의 데이터가 정확한지와 mse를 구한다.                  acc, one\_mse, \_, \_ = self.one\_mse\_acc(self.layer[-1], self.train\_label[iter])                  accuracy += acc                  mse += one\_mse                    # 역전파 시작                  self.back\_propagation(iter)                self.accuracy\_list.append(accuracy / length)              self.mse\_list.append(mse / length)                if i % 100 == 0:                  print("[Epoch] : {} ===> ( [MSE] : {}, [Accuracy] : {} ), ( [test\_MSE] : {}, [test\_acc] : {} )"\                      .format(i, self.mse\_list[i], self.accuracy\_list[i], self.test\_mse\_list[i], self.test\_accuracy\_list[i]))            # 학습 끝난 최종 가중치 전체 weight에 저장          # 최종 가중치로 테스트에 사용하기 위해 저장, file로 넣어 둔다          for i in range(len(self.weight)):              self.weight[i]= self.layer[i].w            # 마지막 back propagation을 통해 업데이트 된 가중치를          # 순전파를 한번더 진행한다.          accuracy = 0          mse = 0          for iter in range(len(self.train\_x\_data)):              self.layer[0].in\_data = self.train\_x\_data[iter]              self.layer[0].out\_data = self.train\_x\_data[iter]                self.foward\_propagation()                acc, one\_mse, \_ , \_= self.one\_mse\_acc(self.layer[-1], self.train\_label[iter])              if acc == 1:                  self.out\_data.append("True")              else:                  self.out\_data.append("False")              accuracy += acc              mse += one\_mse          self.accuracy\_list.append(accuracy / length)          self.mse\_list.append(mse / length)        def foward\_propagation(self):          # 설계한 은닉층에서 출력층까지 돌린다.          # MultiPerceptron 클래스에서 내부적으로 이전 layer를 가지고 있으므로 돌려주기만 하면 된다.          for i in range(1,self.hidden\_layer\_cnt+2):              self.layer[i].neuron()         # 역전파 함수      def back\_propagation(self, iter):              # Chain              # 손실함수 미분              cmp = self.cost\_mse\_prime(self.train\_label[iter], self.layer[-1].out\_data)              # 활성화 함수 미분              sig\_prime = self.find\_active\_prime(self.layer[-1])                # 활성화 함수까지의 Chain 결과              node\_prime = self.node\_prime\_model.copy()              node\_prime[-1] = [cmp[temp\_i] \* sig\_prime[temp\_i] for temp\_i in range(len(cmp))]                weight\_prime = []              # 은닉층 뒤에서 부터 새로운 가중치 저장              for layer\_i in range(self.hidden\_layer\_cnt+1, 0,-1):                  # 역전파의 다음 노드(입력층과 가까운 노드)의 활성화 함수 미분값                  sig\_prime = self.find\_active\_prime(self.layer[layer\_i-1])                    # 가중치 미분값 구하는 파트                  out\_d = np.r\_[self.layer[layer\_i].last\_layer.out\_data, [1]].reshape(-1,1)                  node\_p = np.array(node\_prime[layer\_i]).reshape(1,-1)                  weight\_prime.insert(0, out\_d.dot(node\_p))                    new\_node = self.layer[layer\_i].last\_layer.w.dot(node\_prime[layer\_i])                  node\_prime[layer\_i-1] = new\_node[:-1] \* sig\_prime                # 가중치 갱신              for i in range(len(weight\_prime)):                  self.layer[i].w = self.layer[i].w - self.lr \* weight\_prime[i]          # 하나의 데이터에 대해 정화도와 MSE구한다.      def one\_mse\_acc(self, layer, label):          accuracy = 0          mse = 0            # 순전파 결과에서 가장 높은 값과 라벨의 가장 높은 값이          # 같으면 정확도를 1 높인다.          data\_index = np.argmax(layer.out\_data)          label\_index = np.argmax(label)          if label\_index == -1:              print("[Error] : label에 1 존재하지 X")          elif data\_index == label\_index:              accuracy = 1            # MSE 구하는 함수          mse = self.cost\_one\_mse(label, layer.out\_data)          return accuracy, mse, data\_index, label\_index        # 하나의 데이터 MSE를 구하는 함수      def cost\_one\_mse(self, label, y\_pred):          er = (label - y\_pred) \*\* 2          return sum(er)        # 손실함수 미분 함수      def cost\_mse\_prime(self, label, out\_data):          return (-2)\*(label - out\_data)        # 활성 함수를 미분한 값을 가져오는 함수      def find\_active\_prime(self, layer):          # Input ddata          if layer.activation\_param == 'none':              return layer.none(layer.out\_data)          # Rule data          if layer.activation\_param == 'relu':              return layer.relu\_prime(layer.out\_data)          if layer.activation\_param != "sigmoid":              #print("현재 구현 안함, 자동으로 sigmoid 변경")              pass          return layer.sigmoid\_prime(layer.out\_data) |

**[가중치와 모델 layer 초기 세팅]**

|  |
| --- |
| # 가중치 세팅      def set\_weight(self):          # Input          # 입력층 - 은닉1층          # matrix = 입력 특성수 x 은닉1층의 노드 수          if len(self.weight) != 0:              for i in range(self.hidden\_layer\_cnt+1):                  self.weight[i] = np.array(self.weight[i])                  self.layer[i].w = self.weight[i] # 자신의 클래스에 값 저장                return            w =np.random.normal(self.mean,self.mu, size=(self.input\_feature\_cnt + 1,self.layer[1].hidden\_node))          self.weight.append(w) # 설정한 가중치 값 model에 list로 저장          self.layer[0].w = w   # 각 layer(MultiPerceptron 클래스)에 자신의 가중치 저장            # Hidden - Output          # matrix = 은닉층의 노드 x 출력층 클래스 수          # 은닉층만 적용          for i in range(1, self.hidden\_layer\_cnt+1):              w =np.random.normal(self.mean,self.mu, size=(self.layer[i].hidden\_node+1,self.layer[i+1].hidden\_node))              self.layer[i].w = w # 자신의 클래스에 값 저장              self.weight.append(w) # 설정한 가중치 값 model에 list로 저장          def set\_layer(self):          # 입력 층 설정          # 입력층의 노드 = 입력 데이터의 특성 수          input\_data = MultiPerceptron(node=self.input\_feature\_cnt, name="Input\_Layer")          self.layer.append(input\_data)            # 은닉층1: {활성화함수:tanh, node=7}으로 지정          layer1= MultiPerceptron(input\_data, node=4, activation\_param="sigmoid", name="Layer1")          self.layer.append(layer1)            # 출력층: {활성화함수:softmax, node=출력 클래스 갯수}으로 지정          output = MultiPerceptron(layer1, node=self.output\_class\_cnt, activation\_param="sigmoid", name="Output\_Layer")          self.layer.append(output)            # 모든 레이어 - 입력 - 출력          self.hidden\_layer\_cnt = len(self.layer) - 2            # 모델 설계후 각 지정한 노드를 통해 가중치 설정          self.set\_weight()            for layer in self.layer:              self.node\_prime\_model.append([0 for \_ in range(layer.hidden\_node)]) |

**[Test 메소드]**

|  |
| --- |
| # 테스트 데이터를 넣고 돌리는 함수      def test(self):          # 넣은 데이터를 최적의 가중치로 순전파하여          # 결과값을 확인 후 정확도와 mse를 반환한다.          length = len(self.in\_data)          accuracy = 0          mse = 0          for iter in range(len(self.in\_data)):              self.layer[0].in\_data = self.in\_data[iter]              self.layer[0].out\_data = self.in\_data[iter]              # 순전파              self.foward\_propagation()              # 정확도, mse, 예측과 라벨의 인덱스 값              acc, one\_mse, pred\_index, label\_index = self.one\_mse\_acc(self.layer[-1], self.label[iter])              if acc == 1:                  self.out\_data.append("True")              else:                  self.out\_data.append("False")              accuracy += acc              mse += one\_mse              print("데이터: {},\t라벨: {},  예측: {}\t결과: {}".format(self.in\_data[iter], label\_index+1, pred\_index+1, self.out\_data[iter]))          return accuracy / length, mse / length |

# **2. 실습 #2**

**2.0 기초 데이터 정제**

|  |
| --- |
| # main\_dir  main\_dir = 'C:\\Users\\parks\\Desktop\\대학\\4학년 1학기\\ml\\6week'  # 데이터 셋 읽어들입니다.  dataFrame = pd.read\_csv(os.path.join(main\_dir, "NN\_data.csv"))  data = dataFrame.iloc[:,1:] # 첫번째 열의 경우 필요 없어서 제외  data\_x = data.iloc[:,:3].to\_numpy() # 0~2까지 3개의 특성에 따른 데이터들 data\_x로 변환  data\_y = data.iloc[:,3].to\_numpy()  # 마지막 라벨 열 data\_y로 변환  # 데이터의 클래스 갯수 구하는 함수  data\_encode\_label = []  def find\_class\_num(y):      """\_summary\_      Args:          y (numpy.ndarray): 데이터 셋의 Y값      Returns:          list : 서로다른 클래스 리스트 반환      """      class\_list = {}      # y값에서 같은 값만 넣기 위해 남겨둔다.      index = 0      for d in y:          # 클래스 리스트에 기존에 넣은 클래스가 없으면 넣는다.          if d not in class\_list:              class\_list[d] = index              index += 1      return class\_list  def one\_hot\_encode(y):      """\_summary\_      Args:          y (numpy.ndarray): 데이터 셋의 Y값      Returns:          \_type\_: \_description\_      """      class\_list = find\_class\_num(y)        # 클래스의 갯수      cnt\_class = len(class\_list)      print("Class Summary: Total={}".format(cnt\_class))        # 클래스에 따라 리스트를 만들어 준다.      # 라벨이 숫자가 아닐수도 있기에 dictionary로 받습니다.      one\_hot\_label = {}      for i, key in enumerate(class\_list):          print("\tClass{}번: {}".format(i+1, i))          one\_hot\_label[key] = [1 if class\_list[key]==j else 0 for j in range(cnt\_class)]      print("[One-Hot Mapping] ",one\_hot\_label)        for i in range(len(data\_y)):          data\_encode\_label.append(one\_hot\_label[y[i]])      return cnt\_class  cnt\_class = one\_hot\_encode(data\_y) |

**[Train Test 셋 임의로 셔플 후 분리]**

|  |
| --- |
| # 데이터 셋 분리  def split\_data\_set(data, test\_ratio):      # 데이터 갯수의 범위 만큼 인덱스 랜덤으로 섞은 리스트 생성      index = np.random.permutation(len(data))        # 테스트 갯수 구함      test\_size = int(len(data) \* test\_ratio)        # 인덱스를 테스트 셋 갯수와 훈련셋 갯수만큼 분리      test\_index = index[:test\_size]      train\_index = index[test\_size:]        train\_set = []      test\_set = []      # 각각의 인덱스를 리스트에 저장      for i in test\_index:          test\_set.append(data[i])      for i in train\_index:          train\_set.append(data[i])      return np.array(train\_set), np.array(test\_set)  # 데이터 셋 분리를 위해 데이터와 라벨을 합친다.  all\_data = np.c\_[data\_x, np.array(data\_encode\_label)]  # 분리  train\_set, test\_set = split\_data\_set(all\_data, test\_ratio=0.3) |

**2.1 Train set 훈련**

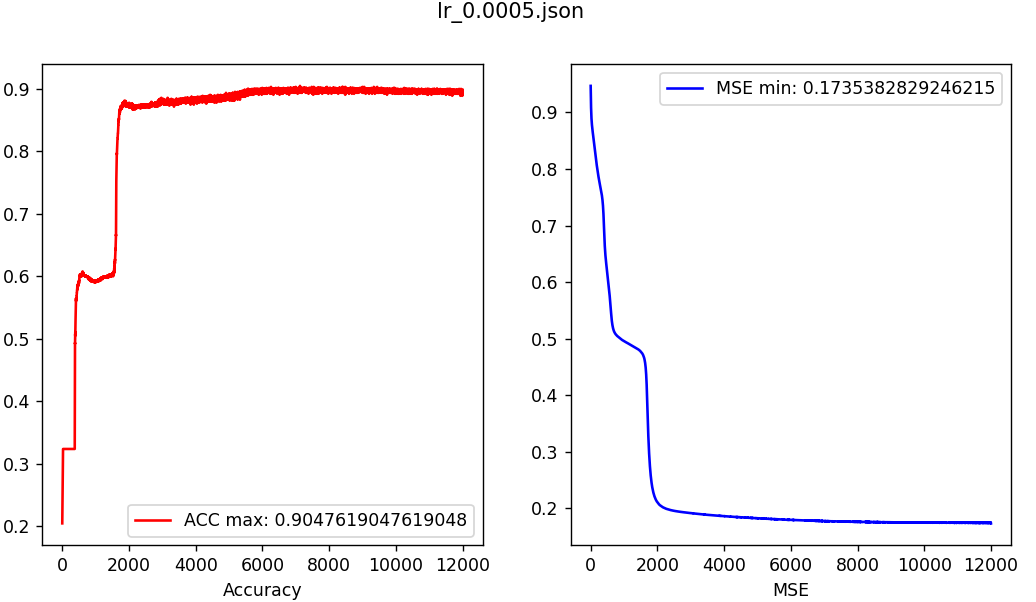
**[Train 시작]**

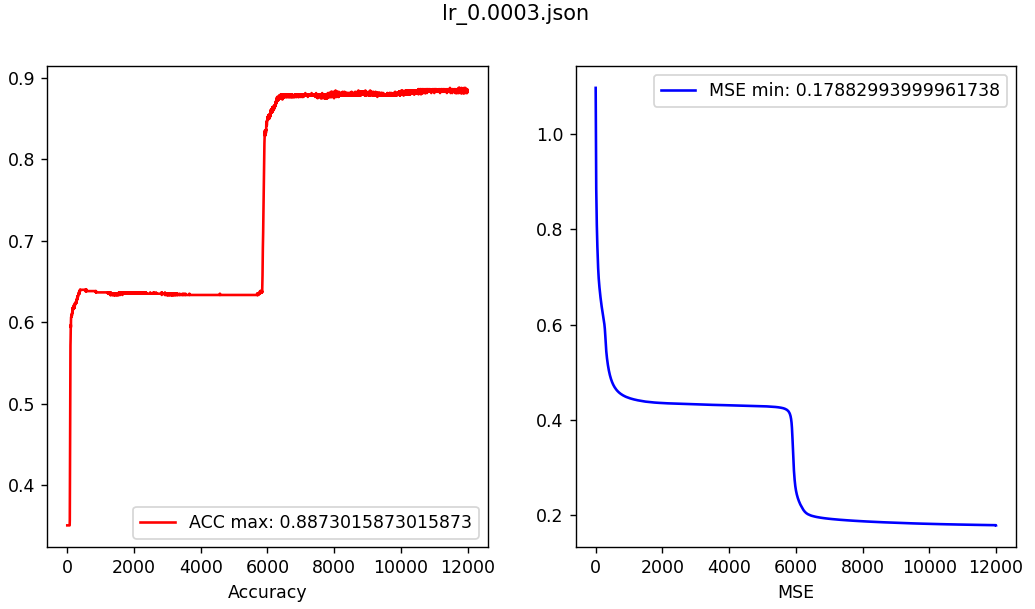
|  |
| --- |
| # train 함수  def train():      # Train      train\_x\_set, train\_label\_set = np.hsplit(train\_set,2)      test\_x\_set, test\_label\_set = np.hsplit(test\_set, 2)        # LR\_list      lr\_list = [0.00001, 0.00005, 0.0001, 0.0003, 0.0005, 0.0007, 0.001, 0.005]      save\_train = list(map(lambda x: x.tolist(),train\_set))      save\_test = list(map(lambda x: x.tolist(),test\_set))        save\_path = os.path.join(main\_dir, "save\_result")      for lr in lr\_list[2:5]:          print("[lr] : ", lr)          # 모델 생성          model = customModel(train\_x\_set, train\_label\_set, epoch=12000, w=[], lr=lr, test\_x=test\_x\_set, test\_y=test\_label\_set)          model.set\_layer() # 계층 설계          model.fit() # 데이터 적용 및 실행            weight = list(map(lambda x: x.tolist(),model.weight))            model\_result = {              'weight': weight,              'accuracy': model.accuracy\_list,              'mse': model.mse\_list,              'test\_acc': model.test\_accuracy\_list,              'test\_mse': model.test\_mse\_list,              'train\_set': save\_train,              'test\_set': save\_test          }            # 결과 값 json 파일로 저장          # 가중치, 정화도, MSE, train 셋,test 셋          with open(os.path.join(save\_path, "lr\_{}.json".format(str(lr))), 'w') as f:              json.dump(model\_result, f) |

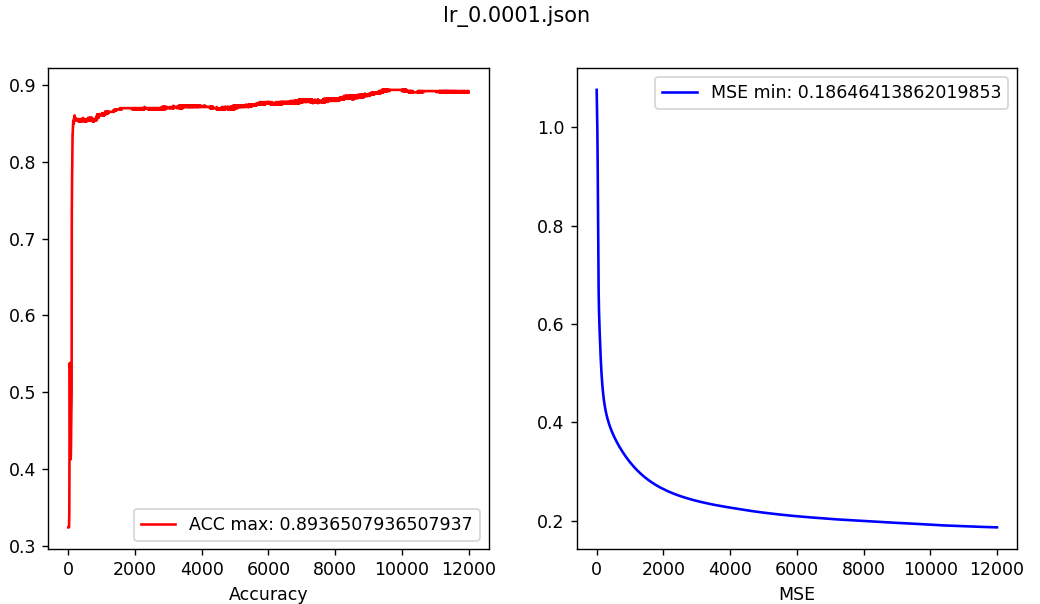
[결과 보는 함수]

|  |
| --- |
| def show():      save\_path = os.path.join(main\_dir, 'save\_result')        save\_file = glob.glob(save\_path+"\\\*.json")      print(save\_file)        for i in range(len(save\_file)):          with open(save\_file[i], 'r') as f:              json\_data = json.load(f)                acc\_list = json\_data['accuracy']              mse\_list = json\_data['mse']                length = len(acc\_list)                plt.figure(figsize=(10,5))              name = save\_file[i].split("\\")[-1]              plt.suptitle(name)              plt.subplot(121)              plt.plot(range(length), acc\_list, c='red', label="ACC max: "+str(max(acc\_list)))              # 테스트 데이터 결과 값 볼려면 아래 주석              # plt.plot(range(len(json\_data['test\_acc'])), json\_data['test\_acc'], c='blue', label="test acc max: "+str(max(json\_data['test\_acc'])))              plt.xlabel("Accuracy")              plt.legend()                plt.subplot(122)              plt.plot(range(length), mse\_list, c='red', label="MSE min: "+str(min(mse\_list)))              # 테스트 데이터 결과 값 볼려면 아래 주석              # plt.plot(range(len(json\_data['test\_mse'])), json\_data['test\_mse'], c='blue', label="test mse max: "+str(max(json\_data['test\_mse'])))              plt.xlabel("MSE")              plt.legend()              plt.show() |

[결과 값]







가장 좋은 결과값 lr = 0.0005

**[Lr = 0.0001]**

텍스트, 컴퓨터이(가) 표시된 사진

자동 생성된 설명

**[Lr = 0.0003]**

텍스트, 서있는, 컴퓨터이(가) 표시된 사진

자동 생성된 설명

**[Lr = 0.0005]**

텍스트, 컴퓨터이(가) 표시된 사진

자동 생성된 설명

# **3. 실습 #3**

**3.1 Test 데이터 결과 출력**

[테스트 함수]

|  |
| --- |
| def test\_data():      # 최종 lr = 0.005      lr = 0.0005      # 값 저장한 파일에서 데이터 읽어온다.      save\_path = os.path.join(main\_dir, 'save\_result')      with open(os.path.join(save\_path,"lr\_{}.json".format(lr)), 'r') as f:          json\_data = json.load(f)          weight = json\_data["weight"]          test\_set = np.array(json\_data["test\_set"])        test\_x\_set, test\_label = np.hsplit(test\_set, 2)      # 테스트 데이터 넣고 반복      model = customModel(test\_x\_set, test\_label, w=weight,epoch=20000, lr=lr)      model.set\_layer() # 계층 설계      acc, mse = model.test() # 데이터 적용 및 실행        print("최종 결과 : Acc={}, MSE={}".format(acc,mse)) |

텍스트, 명판이(가) 표시된 사진

자동 생성된 설명