표지판, 시계이(가) 표시된 사진

자동 생성된 설명

2020년 2학기

NN과제

|  |
| --- |
| 과목: 데이터사이언스개론 |
| 담당교수: 신효섭 |
| 제출일: 2020/11/29 |
| 학과: 기술경영학과 |
| 학번: 201512082 |
| 성명: 백 승 호 |

|  |
| --- |
| [ 목 차 ]  1. 코드  2. 결과  3. 최적의 모델 선정 방법  3.1 데이터 샘플 생성 방법  3.2 데이터 샘플의 개수  3.3 epoch 수  3.4 hidden layer 수  3.5 각 hidden layer 의 neuron 수  3.6 activation 함수 선정  4. 가장 좋은 스펙과 이유 |

**1. 코드**

|  |
| --- |
| ﻿#!/usr/bin/env python3  # -\*- coding: utf-8 -\*-  """  Created on Wed Nov 25 14:53:55 2020  @author: baegseungho  """  import numpy as np  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Input, Dense  from tensorflow.keras import initializers  # y = a + b\*x1 + c\*x2^2 + d\*x3^3 + e 찾아낼수있을까?  def gen\_sequential\_model():  model = Sequential([  Input(3, name='input\_layer'),  # 입력이 x1 x2, x3 세개의 변수로부터 값을 받으니까 3, input layer의 이름은 input\_layer  Dense(16, activation='sigmoid', name='hidden\_layer1', kernel\_initializer=initializers.RandomNormal(mean=0.0, stddev=0.05, seed=42)),  # input layer로부터 히든레이어 구성할거임. 케라스에선 Dense라는 함수로 구성할수있음  # 히든레이어 하나만 만들고 히든레이어안에 16개 뉴런을 쓸거고 액티베이션함수는 히든레이어에선 sigmoid쓰겠ㄱ다.  Dense(16, activation='sigmoid', name='hidden\_layer2', kernel\_initializer=initializers.RandomNormal(mean=0.0, stddev=0.05, seed=42)),  Dense(16, activation='sigmoid', name='hidden\_layer3', kernel\_initializer=initializers.RandomNormal(mean=0.0, stddev=0.05, seed=42)),  Dense(1, activation='relu', name='output\_layer',kernel\_initializer=initializers.RandomNormal(mean=0.0, stddev=0.05, seed=42))  ])  # input layer부터 output layer까지 시퀀스형태로 나열    model.summary()      print(model.layers[0].get\_weights())  # 히든레이어의 가중치들  print(model.layers[1].get\_weights())  print(model.layers[2].get\_weights())  # 아웃풋레이어의 가중치들 프린트!    model.compile(optimizer='sgd', loss='mse')  # sgd = stochastic gradient descent, mse = mean square error  # regression이기때문에 mse쓸거고, 클래시피케이션이라면 cross entropy같은거 쓸거임..  # 이렇게 옵티마이저랑 로스함수 지정해주고 돌리게됨  return model  # y = a + bx1 + cx2^2 + dx3^3 + e 찾아낼수있을까?  def gen\_linear\_regression\_dataset(numofsamples=50, a=1, b=3, c=5, d=10, e=20):  # 실험을 위한 데이터셋 y만들기  np.random.seed(42)  X = np.random.rand(numofsamples, 3)  T = X.transpose()  x1 = T[0]  x2 = T[1]  x2 = np.power(x2, 2)  x3 = T[2]  x3 = np.power(x3, 3)  T = np.vstack([x1,x2,x3])  T = T.transpose()  #0 ~ 1사이의 값을 줌. 샘플의 개수, n차원 어레이. x1과 x2 x3값을 만들어야하니까..3차원으로 한쌍씩 아무거나 생성하면됨  coef = np.array([b, c, d])  # 함수인자로 주는 b, c, d를 계수로보고 a + e를 bias로 봄  bias = a + e  y = np.matmul(T, coef.transpose()) + bias  # for loop안돌기위해 벡터연산! matmul()  return X, y  # 학습 history를 플랏으로 그려보자  def plot\_loss\_curve(history):  import matplotlib.pyplot as plt  plt.figure(figsize=(15,10))  # 그래프 크기 15\*10  plt.plot(history.history['loss'][1:])  # training data에대한 loss  plt.plot(history.history['val\_loss'][1:])  # validation data에대한 loss  plt.title('model loss')  plt.ylabel('loss')  # loss어케되는지보자  plt.xlabel('epoch')  # epoch쭉 증가  plt.legend(['train','test'], loc='upper right')  plt.show()    # 만들어진 모델이용해서 새로운 데이터셋 만든거 넣어보자!  # def predict\_new\_sample(model, x, a=1, b=3, c=5, d=10, e=20):  def predict\_new\_sample(model, x, a=1, b=3, c=5, d=10, e=20):  x = x.reshape(1,3)  y\_pred = model.predict(x)[0][0]  y\_actual = a + b\*x[0][0] + c\*pow(x[0][1], 2) + d\*pow(x[0][2], 3) + e  print("y actual value = ", y\_actual)  print("y predicted value = ", y\_pred)    model = gen\_sequential\_model()  # 모델만들고  T, y = gen\_linear\_regression\_dataset(numofsamples=10000)  # 트레이닝데이터셋 만들고  history = model.fit(T, y, epochs=400, verbose=2, validation\_split=0.3)  # 30번 반복. 실제 트레이닝 각 단계에서의 로스값, 주어진 데이터 자체적으로 train과 validataion스플릿할수있게끔 validation\_split=0.3 => training에 70퍼 validate 하는데 30퍼  plot\_loss\_curve(history)  # weight값이 초기화가 잘못된값이 들어가면 로스값 엄청커져..  # 로컬미니마에 빠지는경우가 있음..  # 어떻게 피하지? 매번돌릴때마다 값이 달라  # 패러미터를 초기화를 고정시켜주는거. 아까 파라미터 65개라그랬음  # Dense에서 kernel\_initializer 라는걸 사용하면됨.  # Dense(1, activation='relu', name='output\_layer',kernel\_initializer=initializers.RandomNormal(mean=0.0, stddev=0.05), seed=42)  print("train loss=", history.history['loss'][-1])  print("test loss=", history.history['val\_loss'][-1])  # 마지막 로스값 출력(train, test)  # 데이터 개수가 많으면 만들수록 로스값 적어지넹  predict\_new\_sample(model, np.array([0.1, 0.3, 0.5]))  # 뉴럴네트워크는 어떤 머신러닝 문제든 풀수있어  # 데이터 충분히주면 학습을 하기때문 |

**2. 결과**

a=1, b=3, c=5, d=10, e=20

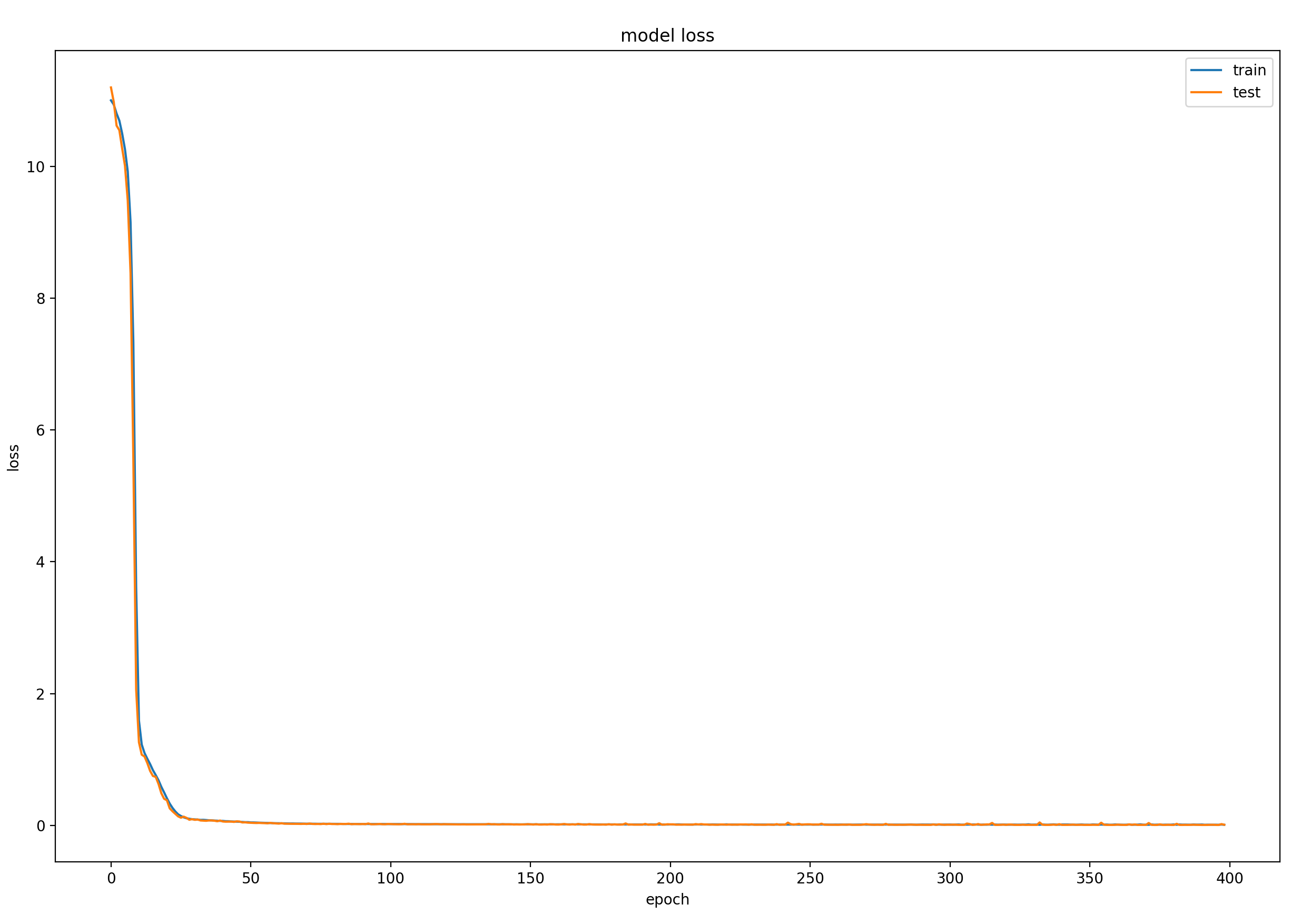
임의 (x1,x2,x3) = (0.1, 0.3, 0.5)

Train Loss : 0.00725

Test Loss : 0.00805

y actual value : 23.0

y predicted value : 23.09895



**3. 최적의 모델 선정 방법**

**3.1 데이터 샘플 생성 방법**

**X:** 데이터 샘플 생성시 랜덤하게 3개의 요소 (x1, x2, x3)를 데이터 생성 모듈의 입력 값 ‘numofsamples’개수 만큼 Numpy로 생성하도록 했다.

**y:** 생성된 X의 x2는 제곱을, x3은 세제곱을 해준 뒤, 모듈의 입력값 b c d 를 Numpy로 묶어서 (x1,x2^2,x3^3)와 (b,c,d)의 병렬 곱 연산(matmul()사용) 후, bias로 모듈의 입력 값 ‘a, e’ 를 토대로 a+e 로 구성한 후 더해줘서 최종적으로 Numpy y를 계산했다.

**범했던 실수:** 입력샘플 X를 리턴할때 (x1, x2, x3)를 리턴하지 않고 (x1, x2^2, x3^3)을 리턴했다. 이렇게 하면 로스는 매우 작게 나와서 학습은 잘 되는것으로 보이지만 실제로 임의의 데이터를 넣어 테스트해보면 큰 차이를 보이게된다. 생각해보면 (x1, x2^2, x3^3)은 y의 방정식을 구성하는 요소일 뿐, 우리가 학습시켜야 할 입력값은 그대로 X가 들어가야 함을 알 수 있다.

**3.2 데이터 샘플의 개수**

구성한 모델은 인풋레이어에 3개의 노드, 3개의 히든레이어 각각에 16개의 뉴런, 아웃풋레이어에 한개의 뉴런으로 구성되어있다.

따라서 파라미터의 개수는 (3\*16 + 16) + (16\*16 + 16) + (16\*16 + 16) + (16\*1 + 1) = 625개 이고, 따라서 최소 10배에 해당하는 6250개 이상의 데이터가 필요하다. 따라서 충분한 데이터를 확보하기 위해 **10000개의 데이터를 생성할 수 있도록 구현**하였다.

**3.3 epoch 수**

epoch 수는 직접 여러 번 학습을 시켜보며 loss값이 충분히 수렴될 수 있을 만큼 설정해주었다. epoch를 늘릴수록 loss값이 줄어들긴 하지만 **학습에 필요한 시간도 그만큼 늘어나게 되므로, 적당하다고 생각되는 만큼인 400회로 구현**하였다.

**3.4 hidden layer 수**

hidden layer 수는 **3개를 넘어가자** 수렴되기 위한 학습에 요구되는 epoch수가 증가하며 **Overffiting되는 모습**을 보였다. 따라서 Hidden layer 수는 적당하다고 생각되는 3개로 구현하였다.

**3.5 각 hidden layer 의 neuron 수**

각 히든레이어의 뉴런의 수는 적당하다고 생각되는 16개로 모두 동일하게 구현하였다.

**3.6 activation 함수 선정**

비선형적인 요소 확보위해 Activation function을 두게되는데, **히든레이어의 activation은 모두 sigmoid로, 아웃풋레이어의 activation은 relu로 구현**하였다.

**4. 가장 좋은 스펙과 이유**

**[ 모델 스펙 ]**

|  |  |  |
| --- | --- | --- |
| **항목** | **스펙** | **이유** |
| Input Layer | 3 인풋 Input layer | x1, x2, x3입력을 받기 때문 |
| Hidden Layer(1~3) | 16 뉴런, activation function: sigmoid | 오버피팅이 발생하지 않는 선으로 3개의 히든레이어로 구성. |
| Output Layer | 1 아웃풋 output layer, activation function: relu | Training이 빠르고 효율적이며 표현력이 좋은 relu사용 |
| 초기화 | Hidden과 Output layer의 초기 파라미터 고정 | 초기화시 가중치에 잘못된 값이 들어가면 로스값이 엄청 커지기 때문에 로컬미니마에 빠질수있음. 따라서 이를 피하기 위해 값을 고정시켜주는 초기화 함수 사용 |
| Loss function | Mean Square Error | Classification이 아닌 Regression이기때문에 사용 |

**[ 기타 스펙 ]**

|  |  |  |
| --- | --- | --- |
| **항목** | **스펙** | **이유** |
| 입력 데이터 | Numpy | 병렬 연산을 통해 훨씬 더 효율적인 연산수행가능 |
| 샘플 데이터 수 | 10000개 | 파라미터 개수에 비례하여 최소 10배이상의 데이터를 마련 |
| Epoch | 400회 | 학습에 필요한 시간과 loss정도를 고려하여 효율적이라고 생각되는 만큼 정했음. |