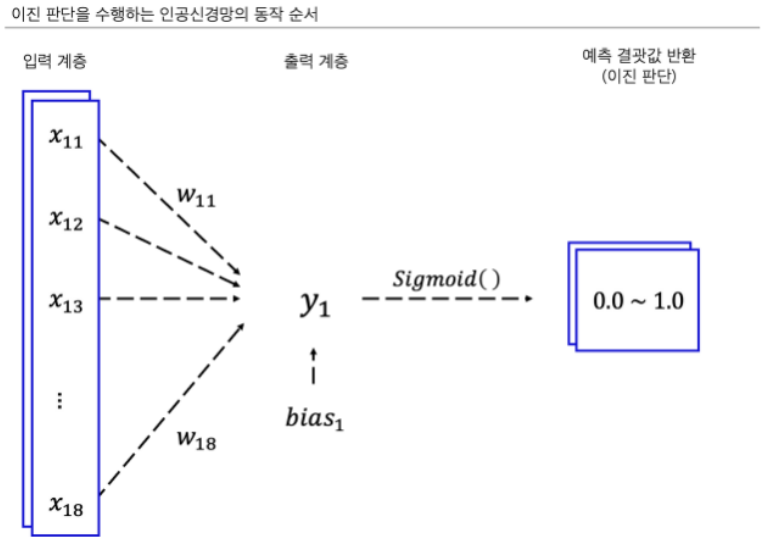
이진 판단 인공신경망(순전파) 프로그래밍

이름 : AIB_15_김동규B
CP1 기간 : 2022.01.04 ~ 2022.01.12

목차

1. 프로젝트 개요



가. 목표

- AI 모델에 대하여 순전파 연산 방식 이해 및 코드 구현
- python을 활용한 0 또는 1로 이진 분류를 진행하는 인공신경망 구성
- 역전파 기능을 제외한 순전파와 성능 평가(정확도 및 손실)만 수행 가능하도록 설계

2. 프로젝트 구성

가. 실습 환경 및 활용 라이브러리

- 프로그래밍 언어 : Python
- 실습 환경 : Google Cloab
- 활용 라이브러리 : Numpy, Pandas, Csv, Matplotlib, python 내장함수

나. 활용 데이터

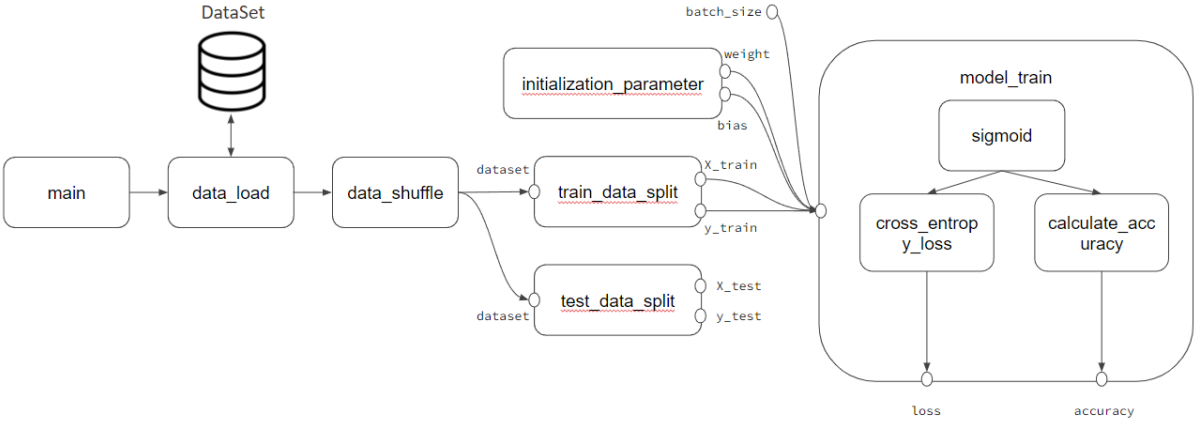
독립변수								
x1	x2	x3	x4	x5	x6	x7	x8	y
136.09375	51.69100464	-0.045908926	-0.271816393	9.342809365	38.09639955	4.345438138	18.67364854	0
99.3671875	41.57220208	1.547196967	4.154106043	27.55518395	61.71901588	2.20880796	3.662680136	1
100.890625	51.89039446	0.627486528	-0.026497802	3.883779264	23.04526673	6.953167635	52.27944038	0
120.5546875	45.54990543	0.282923998	0.419908714	1.358695652	13.07903424	13.31214143	212.5970294	1
121.8828125	53.04267461	0.200520721	-0.282219034	2.116220736	16.58087621	8.947602793	91.01176155	0
125.2109375	51.17519729	0.139851288	-0.385736754	1.147993311	12.41401211	14.06879728	228.1315536	0
141.96875	50.47089779	0.244974491	-0.342664657	2.823578595	16.23818776	8.207743613	85.53258352	0
136.5	49.9327673	0.044623267	-0.374311456	1.555183946	12.81353792	13.31433912	214.813089	0
83.6796875	36.37928102	0.572531753	2.66461052	4.0409699	23.16912864	7.006681423	53.51400467	0
27.765625	28.66604164	5.770087392	37.4190088	73.11287625	62.07021971	1.268206006	1.082920221	1
135.859375	51.93727202	0.065768774	-0.366114187	20.77424749	52.77264803	2.730908619	6.607439551	0
114.28125	41.25396525	0.41182113	0.616996141	2.412207358	20.42794216	9.198391753	88.37057957	0
112.4375	38.2956733	0.501943444	1.07484029	2.81270903	18.13688307	7.859968426	71.29944944	0
23.625	29.94865398	5.688038235	35.98717152	146.5685619	82.39462399	-0.274901598	-1.121848281	1
94.5859375	35.77982308	1.187308683	3.68746932	6.071070234	29.76039993	5.318766827	28.69804799	1
137.2421875	46.45474042	0.045257133	-0.438857507	59.4958194	77.75535652	0.71974817	-1.183162032	0
123.53125	53.34878418	0.072077648	-0.071600995	0.781772575	10.57083301	17.11829958	339.6608262	0
123.46875	45.47508547	0.345780685	0.647414924	32.91973244	65.09419657	1.605538349	0.871363737	1
103.5234375	45.72573893	0.3365333	0.520557925	11.28929766	39.11645317	3.509139254	11.50397981	0
107.9296875	50.58195448	0.320398557	0.277613139	2.022575251	19.80655592	10.47225116	113.0115374	0

종속 변수

- 데이터는 **9개의 열**을 갖고 첫 번째 열 부터 여덟 번째 열(`x1 ~ x8`)은 독립변수, 마지막 열(`y`)은 종속변수를 나타냄
- 데이터는 **21개의 행**을 갖고 첫 번째 행은 각 변수의 이름을 표기하며, 두 번째 행부터 마지막 행은 각 변수의 값을 나타냄
- 출처 : https://drive.google.com/file/d/1SCO0ZGL_EDGWc9Le0JFDw9eww86xk1xJ/view?usp=share_link

3. 프로젝트 수행 절차 및 함수 설명

가. 파이프라인 구성



- 아래 나열된 기능 함수들은 모두 main() 함수에 의해 순차적으로 진행
- 모두 재사용이 가능한 형태인 메서드(함수) 형태로 구현

나. 수행절차 및 구성 함수

1) main() 함수

```
def main():
  data = data_load('/content/binary_dataset.csv')
  data = data_shuffle(data)
  X_train, y_train = train_data_split(data)
  X_test, y_test = test_data_split(data)

  w1, b1 = initialization_parameter(X_train.shape[1])
  accuracy, loss = model_train(X_train, y_train, w1, b1, 4)
  print(f"[Epoch 1] TrainData - Loss = {loss}, Accuracy = {accuracy}")
```

- 순차적으로 data_load, data_shuffle, train_data_split, test_data_split, initialization_parameter, model_train 함수를 호출하며 loss, accuracy 값을 출력한다.

- 데이터 별 행렬 차원

  - data.shape : (20,9)
  - X_train.shape : (16, 8)
  - y_train.shape : (16,)
  - X_test.shape : (4, 8)
  - y_test.shape : (4,)

2) data_load() 함수

```
def data_load(path):
  """
    데이터 불러오는 함수

        Args:
            path (str) : colab 업로드 된 파일 경로

        Retruns:
            data (DataFrame) : binary_dataset.csv
  """
  data = pd.read_csv(path)
  return data
```

- 데이터 불러오는 함수로 Colab 파일에 업로드 된 csv파일의 위치를 path 파라미터로 받아 pandas 라이브러리를 이용해 csv파일을 DataFrame으로 읽어 들인다.

3) data_shuffle() 함수

```
def data_shuffle(data):
  """
    DataFrame 행을 무작위로 섞는 함수

        Args:
            data (DataFrame) : 기존 불러왔던 binary_dataset.csv

        Retruns:
            shuffled_data (DataFrame) : binary_dataset.csv의 행의 순서를 무작위로 섞은 후의 데이터 프레임
  """
  shuffled_data = data.sample(frac=1).reset_index(drop=True)

  return shuffled_data
```

- 입력 파라미터로는 data_load() 함수에서 불러왔던 dataframe을 가지고 행을 무작위로 섞는 함수

- pandas.DataFrame.sample() 개체의 축에서 항목의 임의 샘플을 반환

- 매개변수 frac=1로 지정하여 무작위 추출할 비율을 전체 데이터로 설정하고 reset_index(drop=True)을 통해 기존 인덱스를 버리고 재배열 진행

4) train_data_split() 함수

```
def train_data_split(data, size=0.8):
  """
    학습 데이터 분리 함수

        Args:
            data (DataFrame) : data_shuffle에서 반환 된 데이터 프레임
            size (int) : 학습 데이터 분리 비중 설정

        Retruns:
            X_train (DataFrame) : 독립 변수 y를 제외한 index 0~15에 해당하는 16x8 데이터 프레임
            y_train (Series) : index 0~15에 해당하는 종속 변수 y 값 16개

        Note:
            data.shape[0] : 기존 데이터 프레임의 행의 갯수 20개
            data_train : data[:16] 기존 데이터 index 0~15에 해당하는 값 슬라이싱
  """
```

```
  num = int(data.shape[0]*size)
  data_train = data[:num]
  X_train = data_train.drop(['y'], axis=1)
  y_train = data_train['y']

  return X_train, y_train
```

- 입력 파라미터는 data_shuffle() 함수에서 반환한 dataframe과 학습 데이터 분리 비중으로 이루어져 있고 이를 통해 기존 데이터에서 학습 데이터를 분리하는 함수

- size=0.8로 설정해 놓았기 때문에 기존 데이터의 80%를 학습 데이터로 사용

- 기존 데이터의 0번째 인덱스부터 80%에 해당하는 인덱스 번호까지 슬라이싱 진행

- 슬라이싱이 진행된 data_train에서 종속변수 y을 제외한 x1~x8에 해당하는 독립변수만을 가진 데이터를 X_train으로 설정하고, 종속변수 y에 해당하는 값만 추출하여 y_train으로 설정

- test_data_split() 함수 또한 size=0.2으로 설정한 점을 제외하고 위 방식과 동일하게 테스트 데이터 분리 진행

5) initialization_parameter() 함수

```
def initialization_parameter(inputs):
  """
    가중치 및 편향 생성 함수

        Args:
            inputs (int) : X_train의 열 갯수 -> 독립변수 x1~x8 8개

        Retruns:
            w1 (array) : numpy를 통해 생성된 [0,1]의 범위의 난수를 갖는 8x1 배열
            b1 (array) : 0으로 채워진 1차원 배열
  """
  np.random.seed(40)

  w1 = np.random.randn(inputs,1)
  b1 = np.zeros(1,)

  return w1, b1
```

- 가중치 및 편향 값을 임의로 생성하는 함수

- 프로젝트에서는 input 값을 X_train.shape[1]으로 설정해주었고 이 값은 x1~x8에 해당하는 독립변수의 갯수를 나타냄

- 테스트 시 일관된 값을 도출해내기 위해 random.seed=40으로 고정

- 가중치, 편향 모두 numpy를 통해 무작위 값을 가진 배열 생성

- 생성 예시

  - w1 : [[-0.6075477],[-0.12613641],[-0.68460636],[0.92871475],[-1.84440103],[-0.46700242],[2.29249034],[0.48881005]]
  - b1 : [0]

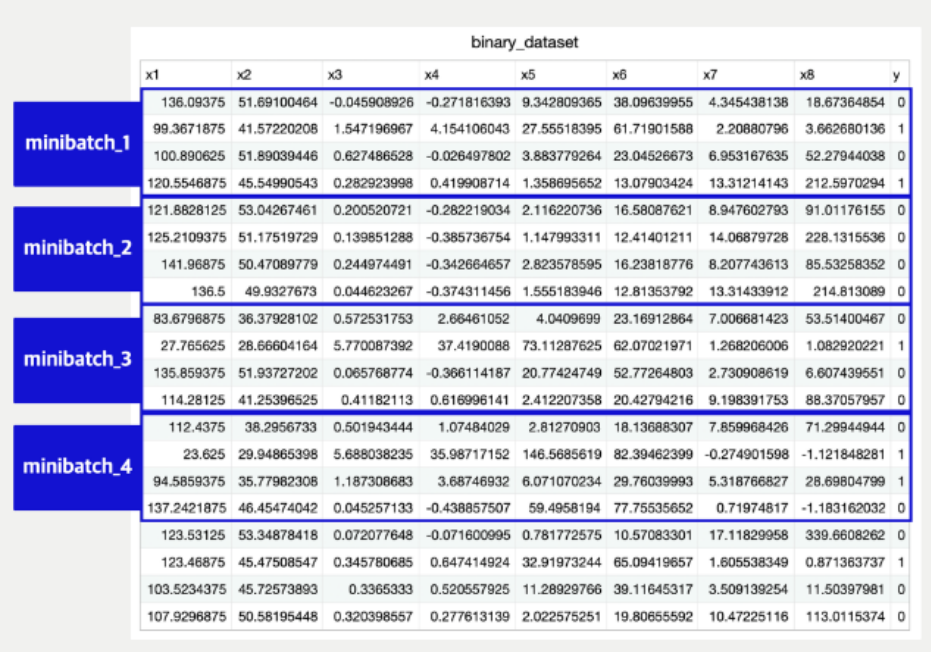
6) model_train() 함수

```
def model_train(X, y, w1, b1, batch_size):
  """
    미니배치를 고려한 학습 데이터 기반의 신경망 연산 및 이진 판단 예측 기능 함수
```

```
        Args:
            X (DataFrame) : 순전파 진행 시 사용 될 독립 변수 데이터 프레임
            y (int) : 순전파 진행 시 사용 될 종속 변수 데이터 프레임
            w1 (array) : initialization_parameter 함수를 통해 생성된 가중치
            b1 (array) : initialization_parameter 함수를 통해 생성된 편향
            batch_size (int) : 사용자 지정 batch_size

        Retruns:
            np.mean(accuracy) (float) : calculate_accuracy 함수를 통해 각 minibatch 마다 계산된 accuracy를 평균 낸 값
            np.mean(cross_entropy) (float) : cross_entropy_loss 함수를 통해 각 minibatch 마다 계산된 loss를 평균 낸 값

        Note:
            a1 : batch_size에 따른 학습 데이터 분리 후 가중치와 내적, 편향 추가
            output : sigmoid 활성화 함수 적용 후 2차원 배열 -> 1차원 배열로 변환 (확률값)
            sigmoid_output : output 중 0.5 이상 값은 1로 이하는 0으로 변환 한 배열
    """
    accuracy = []
    cross_entropy = []

    for i in range(0, len(X), batch_size):
      minibatch = X[i:i+batch_size]
      a1 = np.dot(minibatch, w1) + b1
      output = np.concatenate(sigmoid(a1).tolist())
      sigmoid_output = list(map(lambda x : 1 if x>=0.5 else 0, output))

      accuracy.append(calculate_accuracy(sigmoid_output, y[i:i+batch_size]))
      cross_entropy.append(cross_entropy_loss(output, y[i:i+batch_size]))

    return np.mean(accuracy), np.mean(cross_entropy)
```

- 위 함수를 통해 생성된 학습 데이터(X_train, y_train)와 가중치/편향(w1, b1) 및 사용자 지정 batch_size를 통해 인공신경망(순전파) 연산을 진행하고 이진 판별을 수행하여 정확도 및 손실을 계산하는 함수

- 지정 batch_size 만큼 X_train 데이터를 나누어 가중치 곱의 합 계산 수행



	x1	x2	x3	x4	x5	x6	x7	x8	y
minibatch_1	136.09375	51.69100464	-0.045908926	-0.271816393	9.342809365	38.09639955	4.345438138	18.67364854	0
	99.3671875	41.57220208	1.547196967	4.154106043	27.55518395	61.71901588	2.20880796	3.662680136	1
	100.890625	51.89039446	0.627486528	-0.026497802	3.883779264	23.04526673	6.953167635	52.27944038	0
	120.5546875	45.54990543	0.282923998	0.419908714	1.358695652	13.07903424	13.31214143	212.5970294	1
minibatch_2	121.8828125	53.04267461	0.200520721	-0.282219034	2.116220736	16.58087621	8.947602793	91.01176155	0
	125.2109375	51.17519729	0.139851288	-0.385736754	1.147993311	12.41401211	14.06879728	228.1315536	0
	141.96875	50.47089779	0.244974491	-0.342664657	2.823578595	16.23818776	8.207743613	85.53258352	0
	136.5	49.9327673	0.044623267	-0.374311456	1.555183946	12.81353792	13.31433912	214.813089	0
minibatch_3	83.6796875	36.37928102	0.572531753	2.66461052	4.0409699	23.16912864	7.006681423	53.51400467	0
	27.765625	28.66604164	5.770087392	37.4190088	73.11287625	62.07021971	1.268206006	1.082920221	1
	135.859375	51.93727202	0.065768774	-0.366114187	20.77424749	52.77264803	2.730908619	6.607439551	0
	114.28125	41.25396525	0.41182113	0.616996141	2.412207358	20.42794216	9.198391753	88.37057957	0
minibatch_4	112.4375	38.2956733	0.501943444	1.07484029	2.81270903	18.13688307	7.859968426	71.29944944	0
	23.625	29.94865398	5.688038235	35.98717152	146.5685619	82.39462399	-0.274901598	-1.121848281	1
	94.5859375	35.77982308	1.187308683	3.68746932	6.071070234	29.76039993	5.318766827	28.69804799	1
	137.2421875	46.45474042	0.045257133	-0.438857507	59.4958194	77.75535652	0.71974817	-1.183162032	0
	123.53125	53.34878418	0.072077648	-0.071600995	0.781772575	10.57083301	17.11829958	339.6608262	0
	123.46875	45.47508547	0.345780685	0.647414924	32.91973244	65.09419657	1.605538349	0.871363737	1
	103.5234375	45.72573893	0.3365333	0.520557925	11.28929766	39.11645317	3.509139254	11.50397981	0
	107.9296875	50.58195448	0.320398557	0.277613139	2.022575251	19.80655592	10.47225116	113.0115374	0

- sigmoid 활성화 함수를 통해서 결과값을 0~1사이의 값으로 추출

  - 출력 예시 : [[3.76489635e-020] ,[2.10561055e-130] ,[1.77605941e-041], [1.00000000e+000]]

- output은 sigmoid을 통해 생성된 2차원 배열 값들을 concatenate를 통해 아래 출력 예시처럼 변경

  - 출력 예시 : [3.76489635e-020 ,2.10561055e-130 ,1.77605941e-041, 1.00000000e+000]

- sigmoid_output은 output을 통해 생성된 1차원 배열 값들을 비교하여 0.5 이상인 값은 1, 이하인 값은 0으로 변경하고 모델을 통해 예측한 y값을 배열 형태로 저장

  - 출력 예시 : [0, 0, 0, 1]

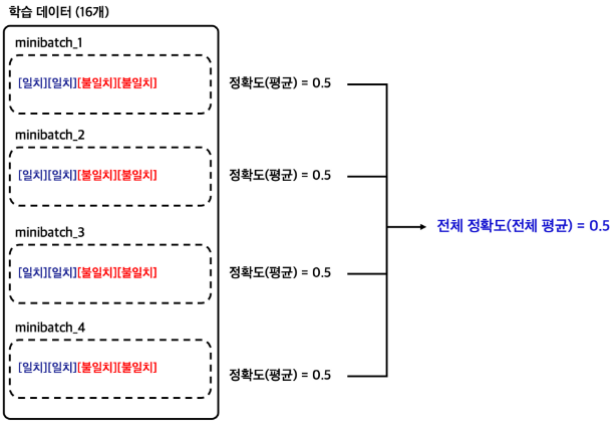
- 인공신경망(순전파)를 통해 생성된 output과 sigmoid_output을 가지고 calculate_accuracy, cross_entropy_loss 함수를 호출하여 accuracy와 loss 값 계산

- batch_size 별 계산된 accuracy와 loss 값을 각 배열에 추가하고 평균 내어 최종 출력

7) calculate_accuracy() 함수

```
def calculate_accuracy(y_pred, y_true):
  """
    정확도 연산 기능 구현 함수

        Args:
            y_pred (list) : batch_size 별 순전파를 진행한 모델의 예측 값(1 또는 0)을 담은 sigmoid_output 배열
            y_true (Series) : batch_size 별 실제 타겟 값

        Retruns:
            minibatch_accuracy (float) : batch_size 별 정확도 평균 값

        Note:
            check_match : y_pred와 y_true의 일치 여부 확인 후 일치하는 수 만큼 cnt 증가
  """
  cnt = 0
  minibatch_accuracy = 0
  check_match = np.equal(y_pred,y_true)

  for correct in check_match:
    if correct:
      cnt+=1

  minibatch_accuracy = cnt/len(y_pred)

  return minibatch_accuracy
```

- model_train()함수에서 calculate_accuracy(sigmoid_output, y[i:i+batch_size]) 형태로 호출되어 모델을 통해 생성된 예측 값을 담은 sigmoid_output 배열과 실제 y값 데이터를 인자로 받아 일치 여부 판별 후 정확도 계산



8) cross_entropy_loss 함수

```
def cross_entropy_loss(y_pred, y_true):
  """
    손실(교차 엔트로피)값 연산 기능 구현 함수

        Args:
            y_pred (array) : batch_size 별 순전파를 진행한 모델의 확률 값을 담은 output 배열
            y_true (Series) : batch_size 별 실제 타겟 값

        Retruns:
            second (float) : batch_size 별 loss 평균 값
  """
  first = y_true * np.log(y_pred + 1e-7)
  second = -np.sum(first)

  return second
```

- model_train()함수에서 cross_entropy_loss(output, y[i:i+batch_size]) 형태로 호출되어 손실(교차 엔트로피)값 연산 기능 구현 함수로 batch_size별 accuracy 계산과 동일하게 진행

4. 프로젝트 수행 결과

```
[Epoch 1] TrainData - Loss = 12.088571713218741, Accuracy = 0.625
```

- main() 메서드가 동작하면서 1 epoch에 따른 손실 값과 정확도 출력
- 역전파 기능이 존재하지 않음으로 2 epoch은 존재하지 않고 손실 값이 매우 높으며 정확도가 높지 않다.
- 단순 1차원적 구조의 인공신경망(순전파) 기능을 구현했기 때문에 모델의 학습을 통한 테스트 데이터의 정확도와 손실을 계산 할 수 없다.

5. 자체 평가 의견

- 프로젝트 자체 결과물은 기획 의도 및 목표에 부합하는 정도가 높다고 생각
- tensorflow / sklearn을 활용하지 않고 인공신경망(순전파)을 코드로 구현했던 점, 파이프 라인을 통해 흐름을 파악했던 점에서 전반적인 연산 방식 및 해당 연산의 필요성을 배웠다.
- train_test_split, sigmoid, cross_entropy 등 내부 동작 원리 및 연산 방식 이해에 도움이 많이 되었다.
- 역전파, 은닉층의 구성이 없는 1차원적 인공신경망 구성과 단순히 예측 결과 값과 실제 정답 데이터의 일치 여부 비교를 통해 정확도를 계산 했던 부분이 아쉬웠다.