AI Programming

Lab 02: DNN1

소속: 컴퓨터정보공학부

학번: 2019202103

이름: 이은비

제출날짜: 231008

텍스트, 로고, 폰트, 등록 상표이(가) 표시된 사진

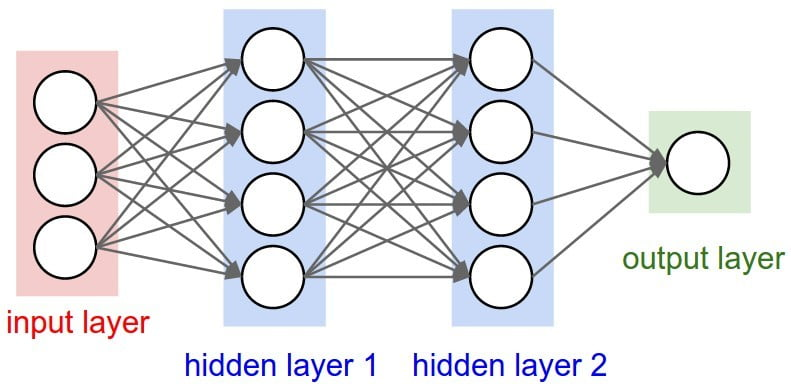
자동 생성된 설명

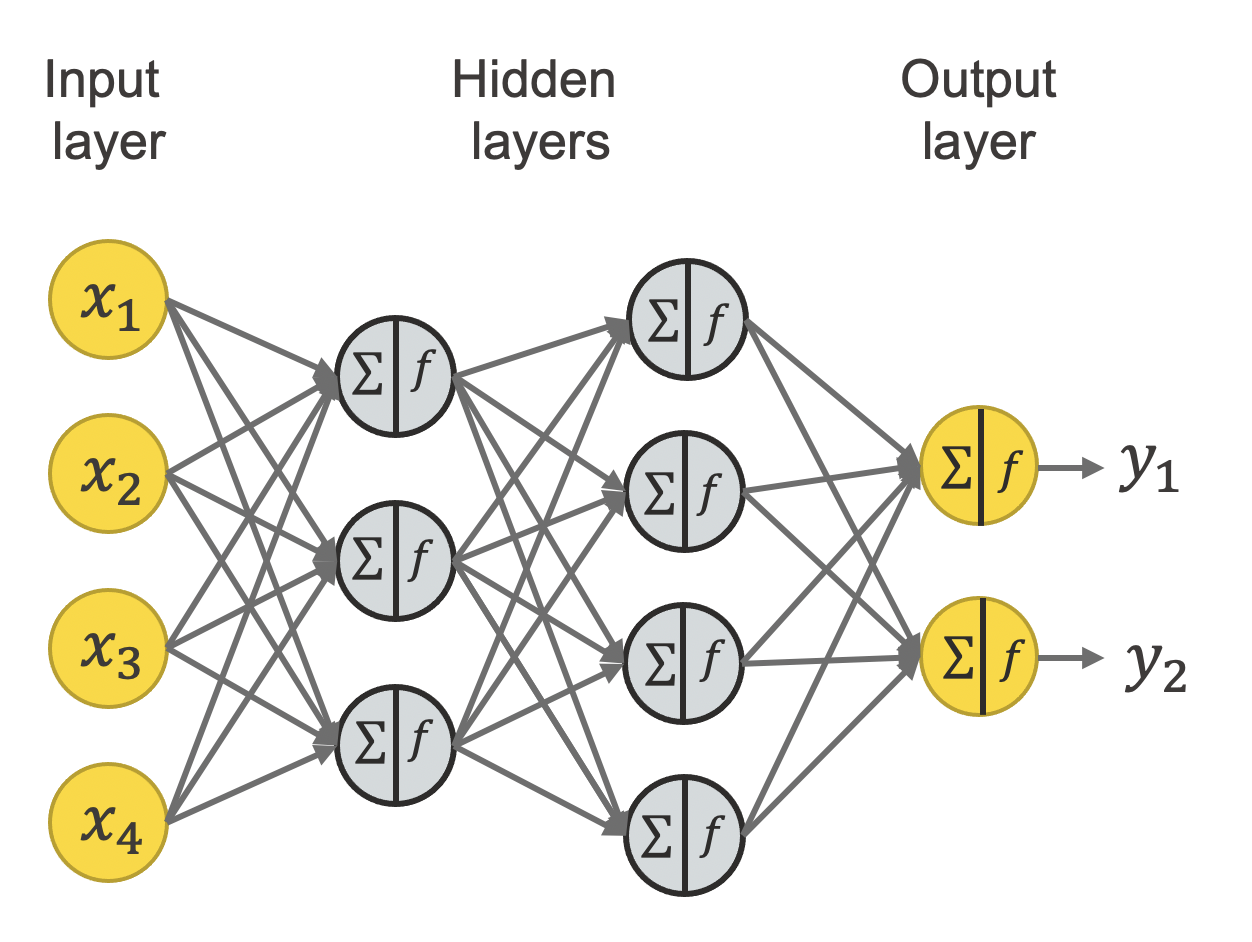
**<Lab Objective>**

Lab 02: Simple Deep Neural Network 실습으로 MNIST dataset을 활용하여 간단한 Neural Network model을 구현합니다. Deep Neural Network는 개략적으로 아래 그림과 같은 구조를 갖고 있다고 볼 수 있는데, 보통 2개 이상의 layer들을 갖고 있는 즉, 간단하게 말하면 어느 정도 복잡도가 있는 신경망을 의미하며 기본적인 forward path와 backward path에서 Linear Regression과 Logistic Regression 즉, linear한 형태의 식과 non-linear한 형태의 식의 조합으로 network를 구성한 후 MNIST data를 활용하여 임의의 inputs data를 넣어서 구성했던 network를 바탕으로 layer에 맞게 식을 재구성하여 loss값을 구하고 그 값에 따른 gradient를 조정하여 이전 weight를 수정하는 backward또한 진행합니다. 그리고 실습을 거친 최종 node에서 구해지는 값들을 통해 실제 loss값과 accuracy까지 구하는 과정을 거칩니다.

그리고 위와 같은 Deep Neural network의 구조의 개념과 함께 그에 쓰이는 function과 수식들을 보면 다음과 같이 정리할 수 있는데, DNN이 단일 class output 에도 적용될 수 있지만 Multi-class output에도 적용될 수 있으며 loss function의 형태가 이에 따라 조금 다를 수 있습니다. forward에서 도출된 결과인 loss값을 반영하여서 backward값을 결정하는데 사용하는 개념에는 Gradient Descent, Chain Rule, Dynamic Programming(Memorization)과 같은 개념이 쓰이는데 각 개념을 간단히 설명하면 error 즉 loss값이 작아질 수 있도록 gradient를 조정하여 그에 해당하는 가중치(weight)를 바꾸는 것이며, ChainRule은 backward로 오는 과정에서 전체식에서 특정변수에 해당하는 변수에 대해 편미분 하는 과정으로 이전 node의 weight를 구하기 위해 사용합니다. Dynamic Programming은 backward path를 진행하기 이전 forward path를 진행할 때 결과로 나왔던 가중치들(weight)를 memorization하는 것입니다.

그리고 사용하는 실습에 사용하는 activation function인 sigmoid 외에 다른 functions로는 tanh,ReLU, LeakyReLU, softplus와 같은 것들이 있으며 사용하는 dataset과 layer의 구조에 따라서 다르게 사용합니다. 그리고 실습에서는 주어진 function과 값들을 이용해서 expected된 결과를 출력할 수 있도록 합니다.

위의 그림은 단일 class

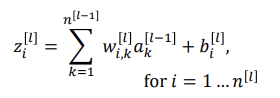


위의 그림은 multi-class

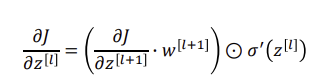
**<Whole simulation program flow>**

MNIST Dataset을 사용하기 용이하게 전처리하는 과정을 거칩니다. 그리고 Utility functions들 즉, forward path에서 만든 sigmoid 와 softmax의 식을 정의합니다. 그리고 식을 활용하기 이전 plt을 통해 dataset이 어떤 형태로 존재하고, dimage의 형태로도 확인합니다. 또한 Model class를 define하여 기본적인 forward path를 진행하고 앞선 과정과 같게 Backpropagation에 쓰이는 activation functions를 정의(dJdz\_sigmoid,dJdz\_softmax) backward path를 진행할 수 있도록 합니다.

실습에서는 expected outputs를 도출해내기 위해서 다음 식(아래 그림,출처: 수업 handouts)을 이용하였습니다.

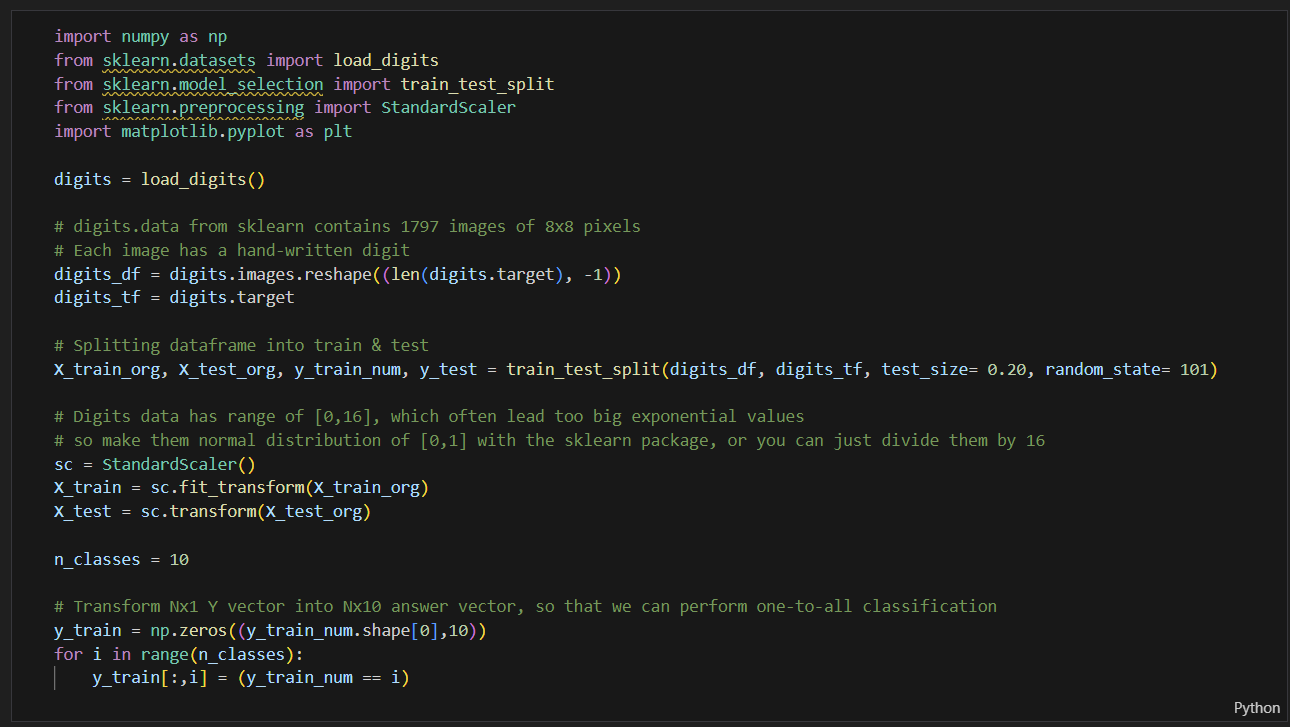


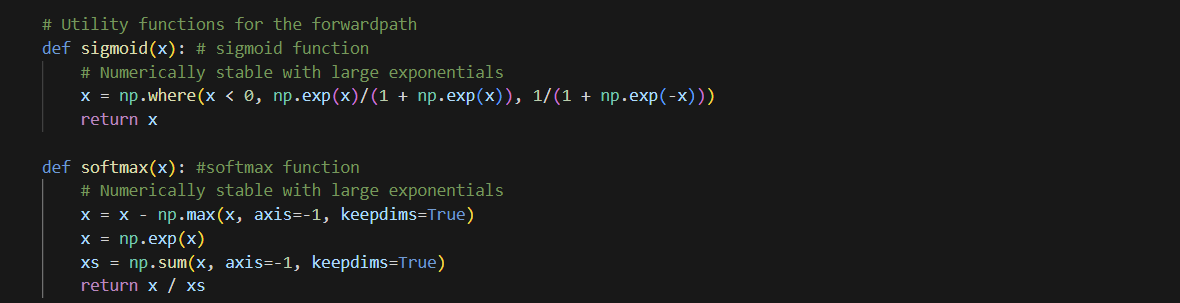
그리고 backpropagation path에 쓰이는 식은 다음과 같습니다.

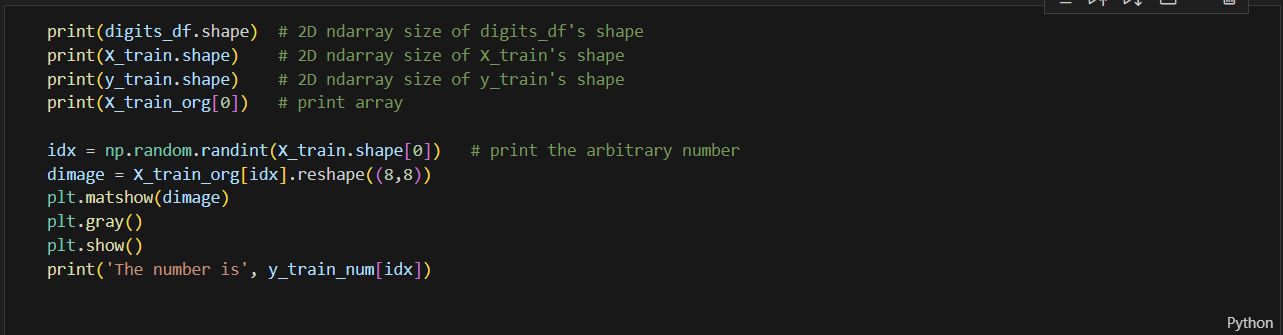


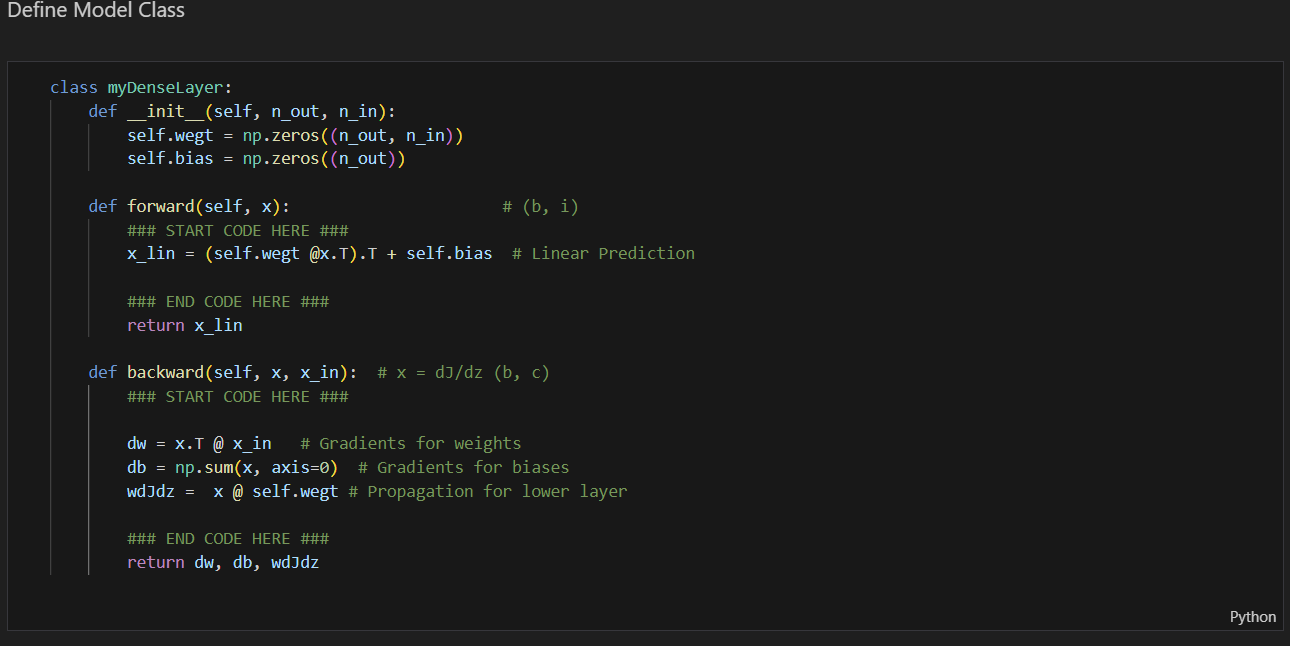
그리고 개념적으로는 forward path에서는 각 input layer에서 hidden layer1으로 갈 때, hidden layer1에서 hidden layer2로, hidden layer에서 output layer로 갈 때 존재하는 노드에서 노드까지 갈 때는 data값과 weight 값을 곱한 linear형태로 다음 layer의 node로 전해지고 전달받은 노드에서는 sigmoid같은 activation function을 이용하여서 non-linear한 형태의 식을 통과한 결과를 node의 data결과 값으로 갖습니다. 위와 같은 형태로 은닉 층. 즉, hidden layer를 통과한 값은 최종적으로 output layer의 형태를 가지며 MSE 혹은 cross entropy와 같은 식을 주어진 layer에 따라서 더 효과적인 식을 선택하여 prediction한 값과 ground truth값을 통과시켜서 최종적인 loss값을 도출해 냅니다. 실습에서는 loss값을 구하는 과정에서 cross entropy를 적용하였으며 epoch 500마다의 loss값을 계산 후 print 하고 epoch를 반복하는 과정에서 loss값을 반영한 backpropagation이 진행되고 그에 따른 layer-by-layer사이의 weights를 update, 그리고 반복되는 과정에서 loss가 줄어들고 accuracy가 증가하는 것을 알 수 있습니다.

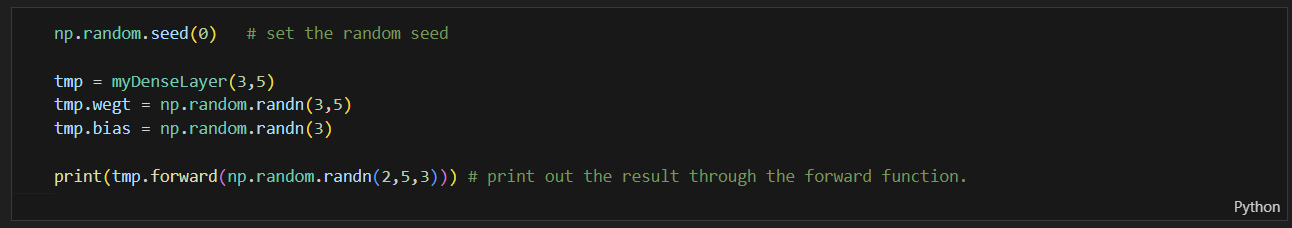
**<Brief comments on each code block (English only)>**

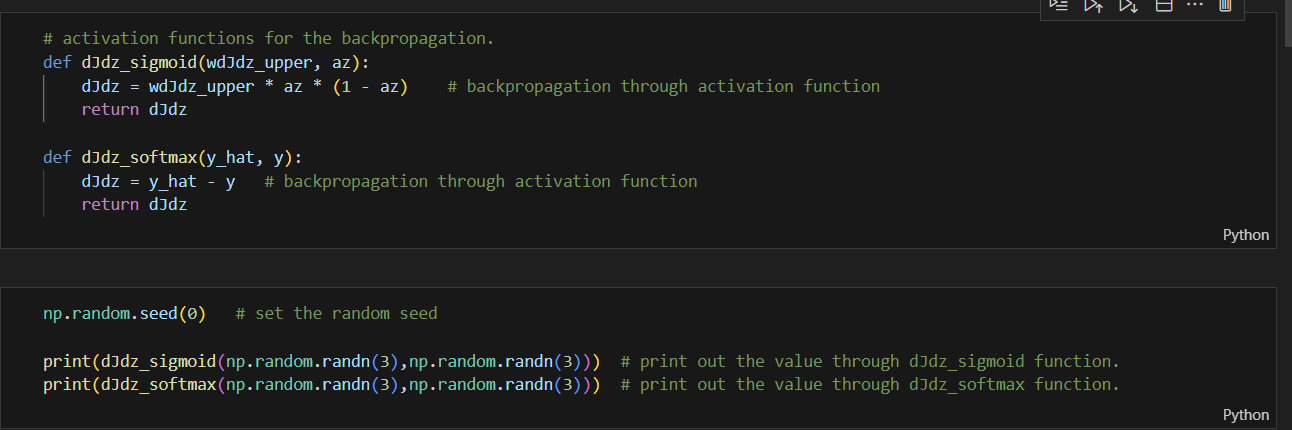
****

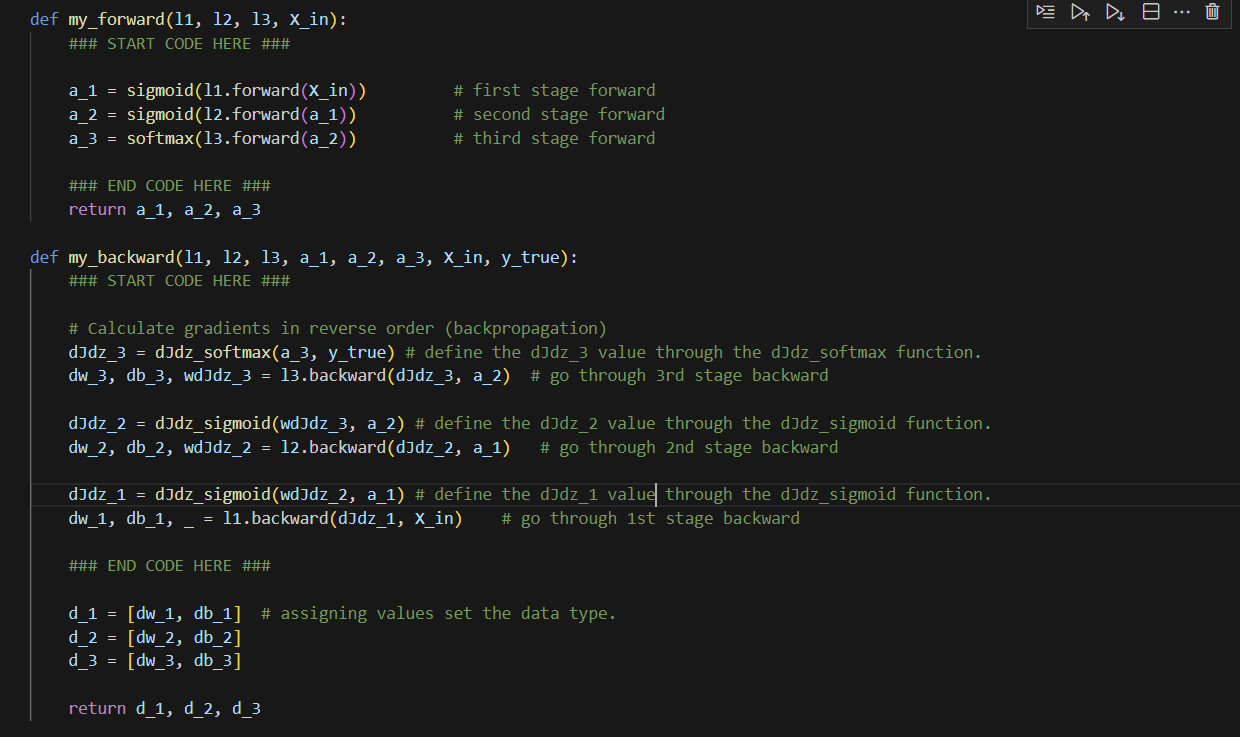
****

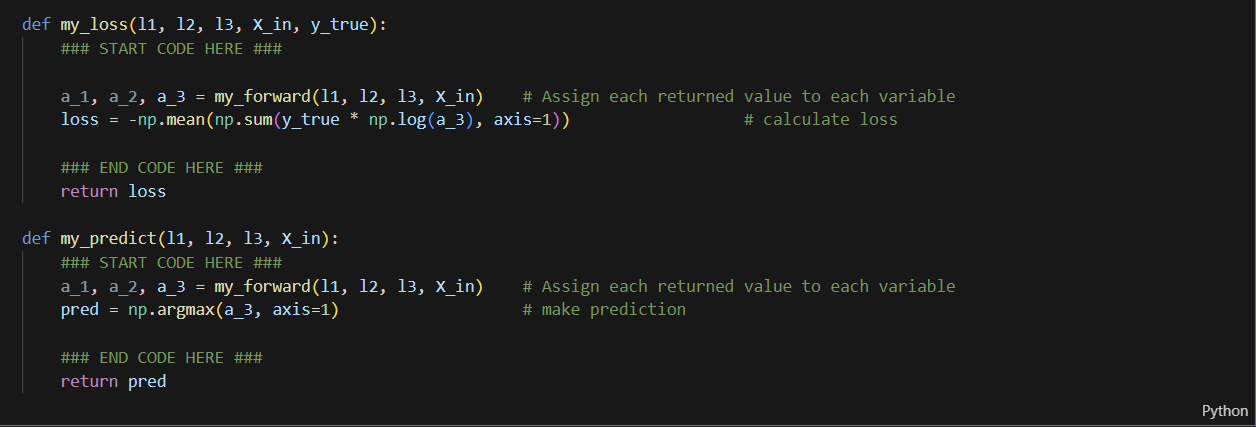
****

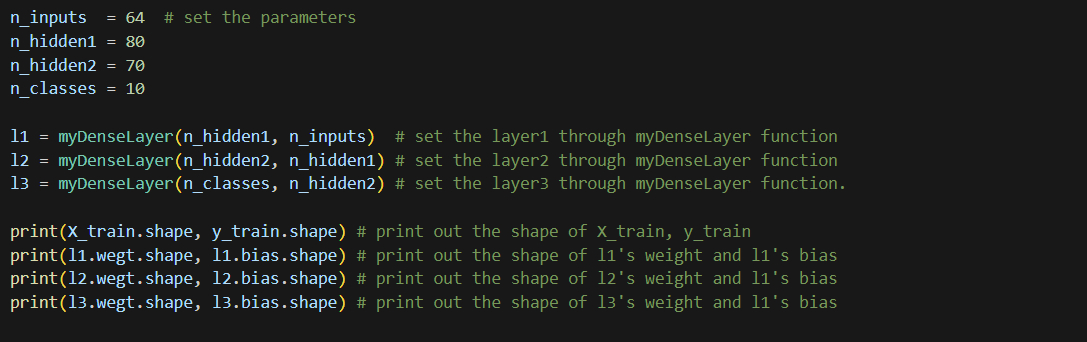
****

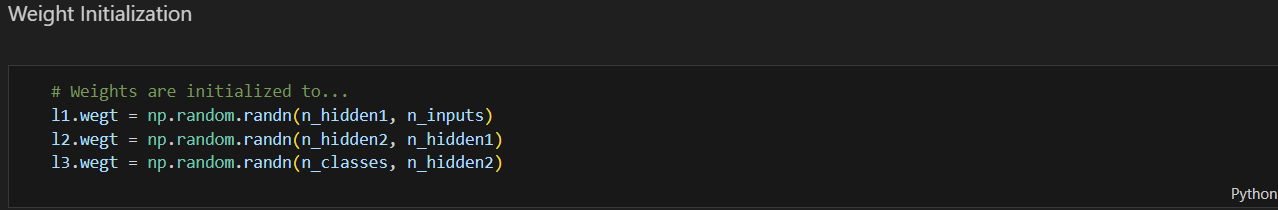
****

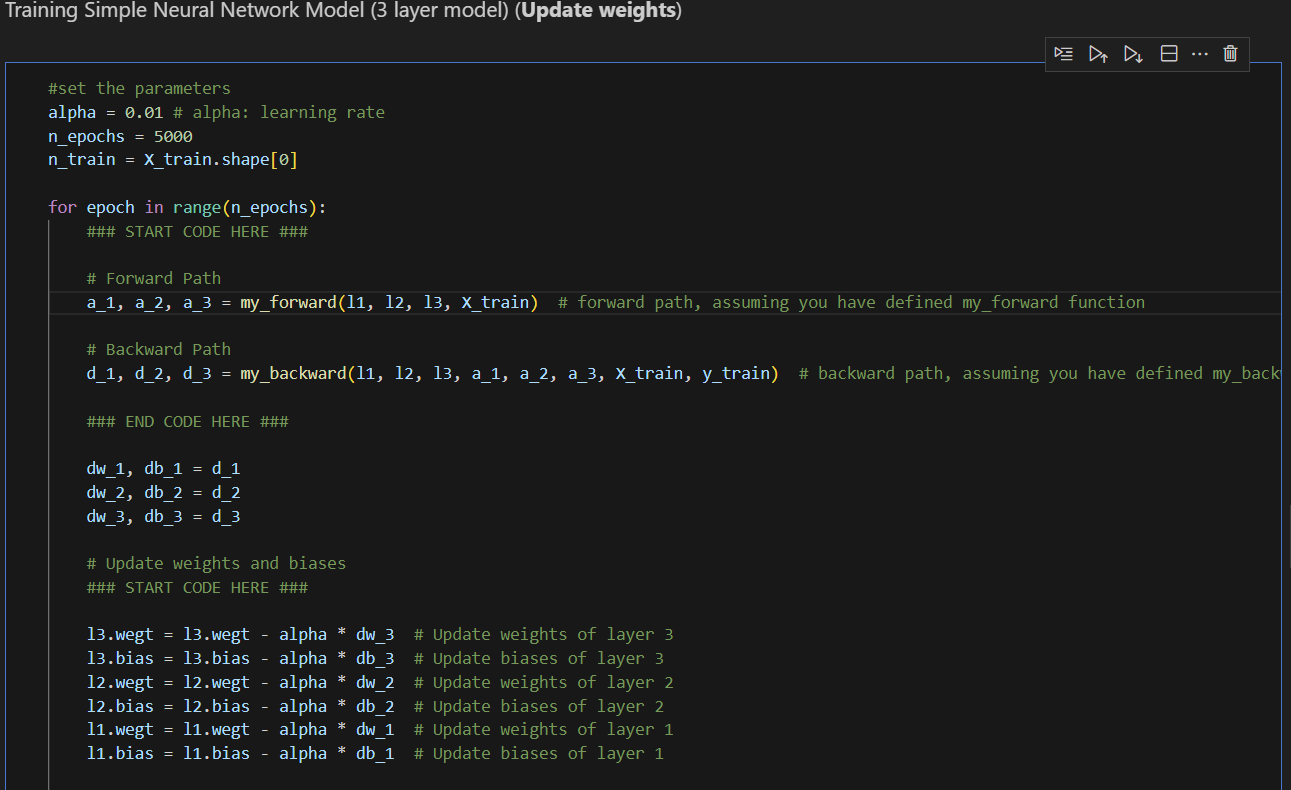
****

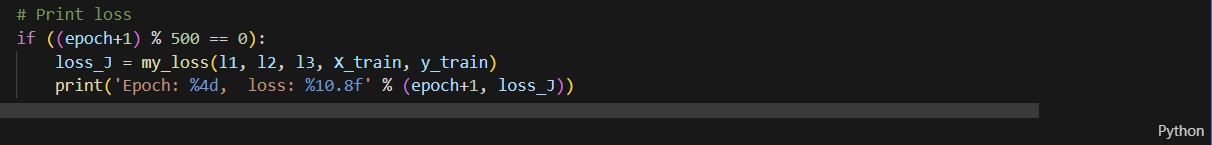
****

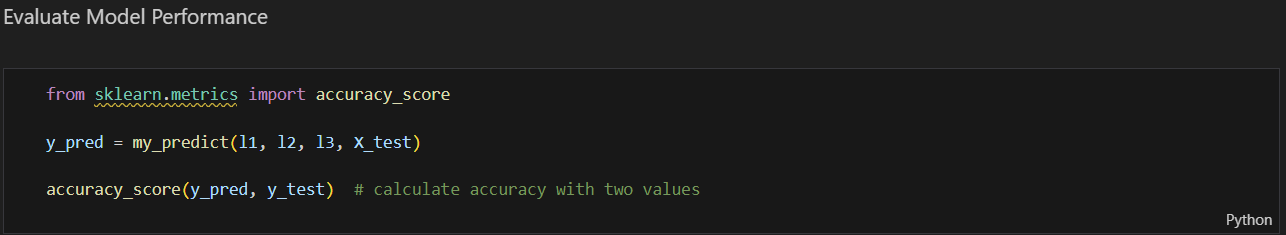
****

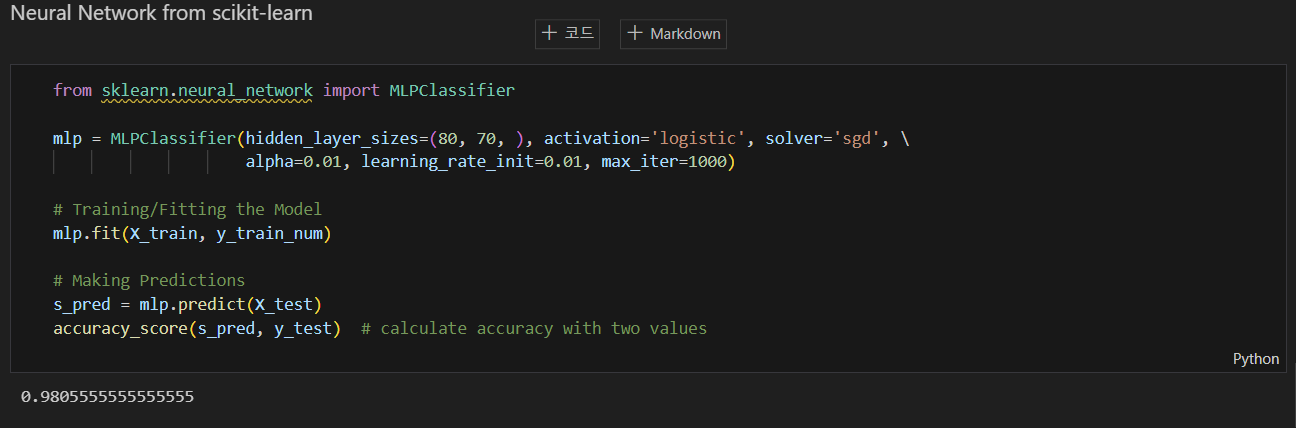
****

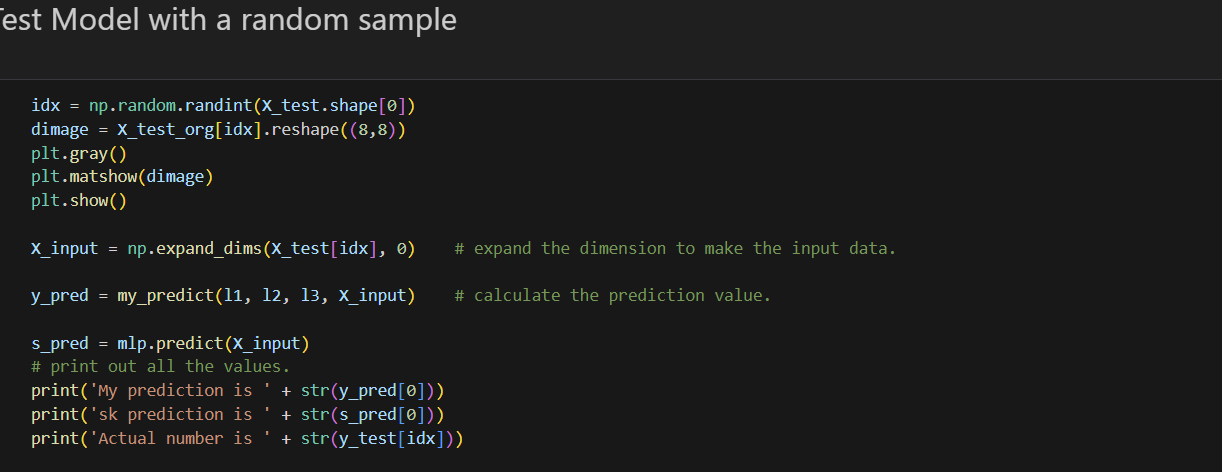
****

****

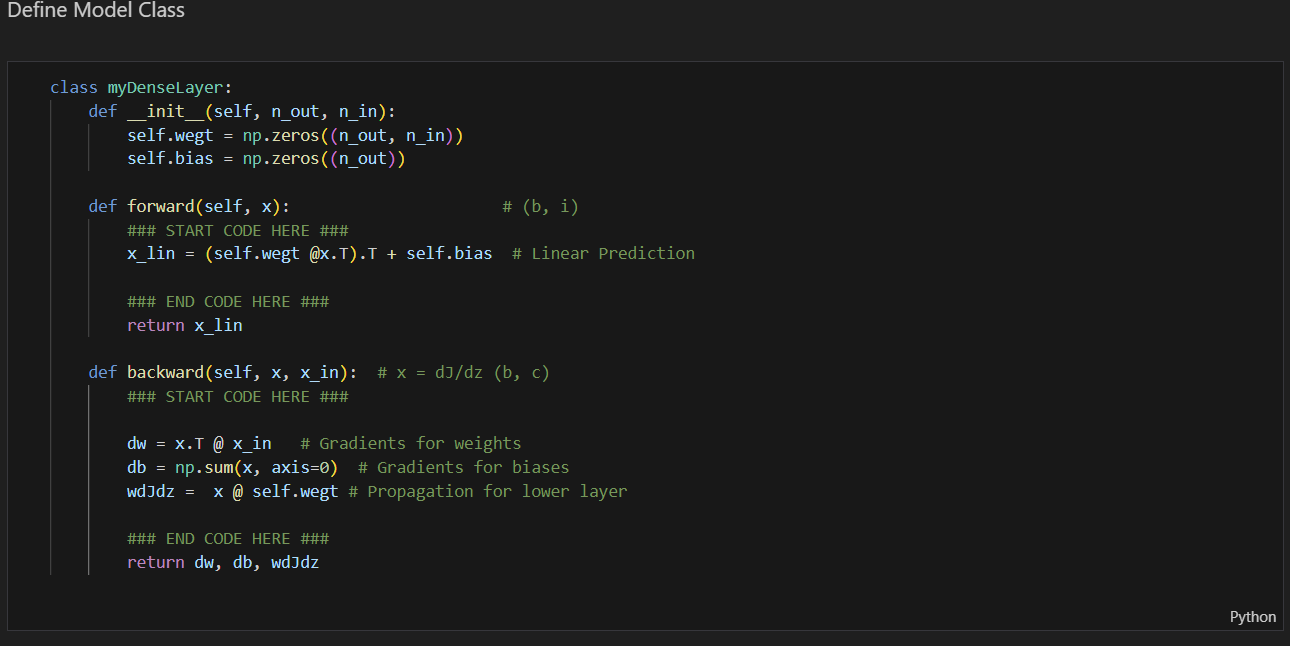
****

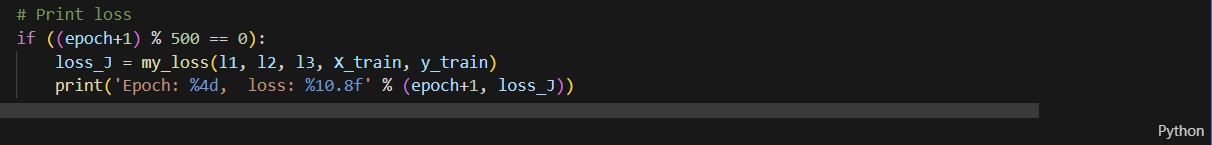
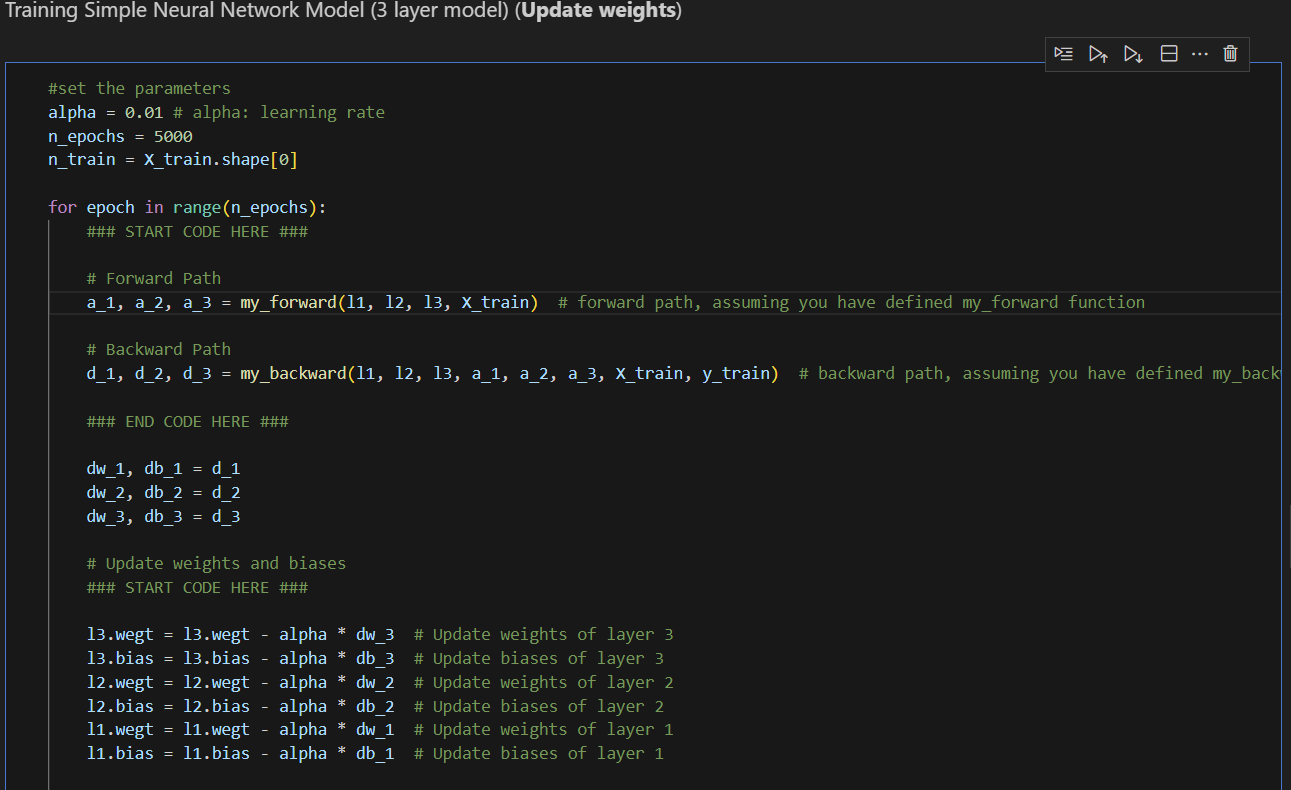
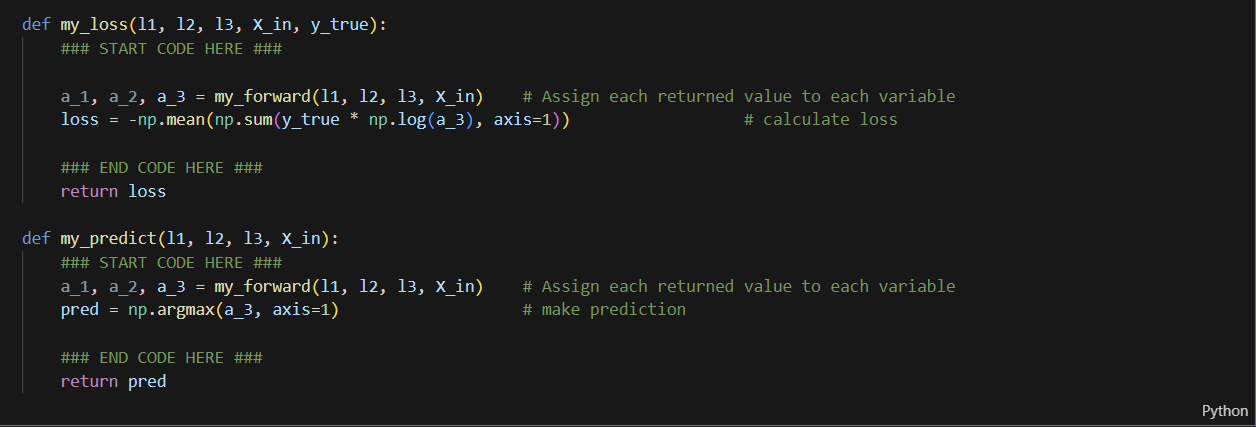
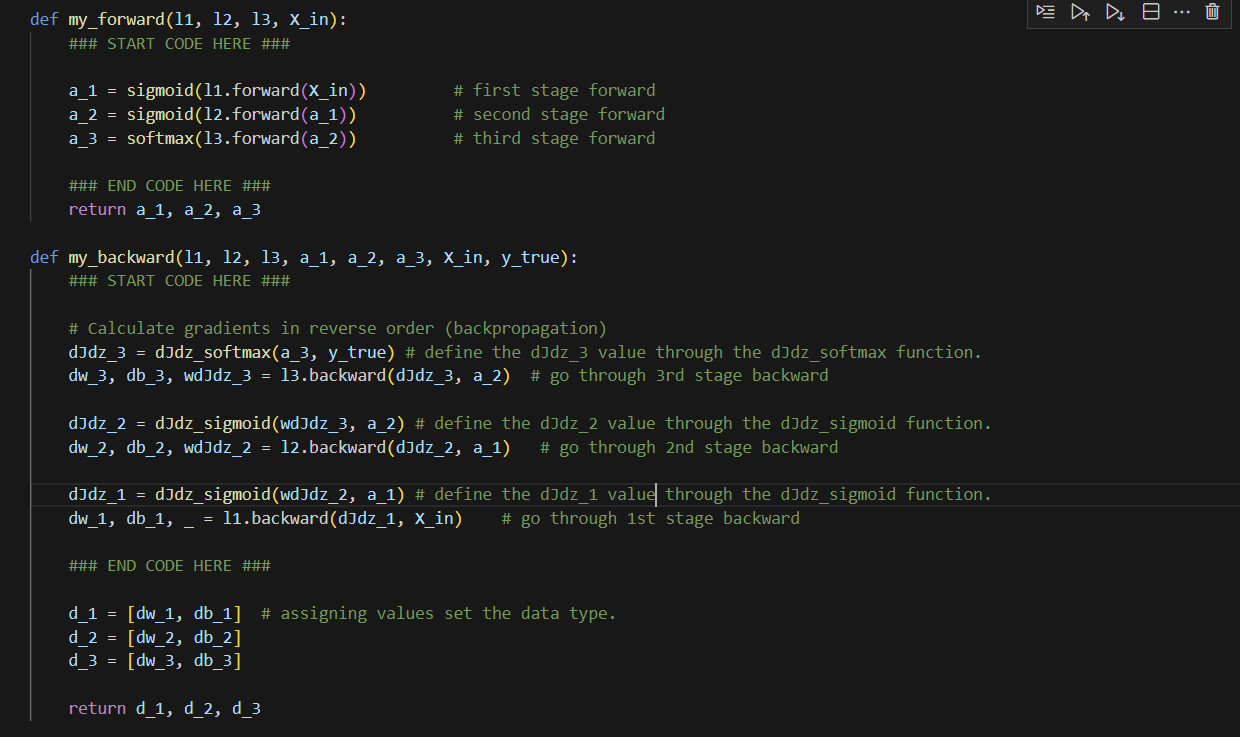
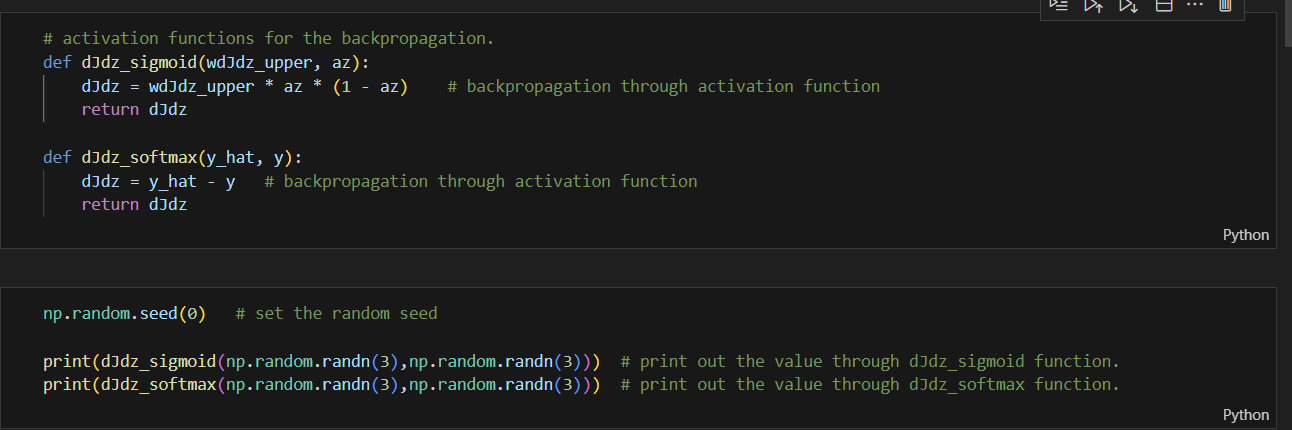
****

****

****

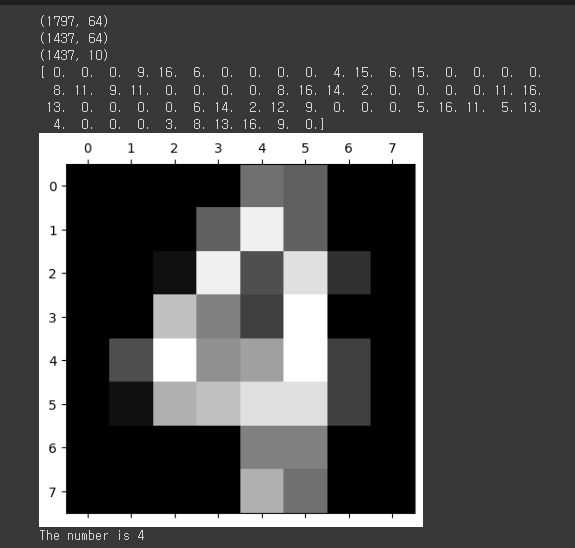
**<Your codes & comments on your codes (English only)>**

****

****

**<Simulation results (screenshots)>**

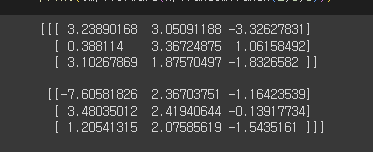
구글 코랩 이용해서 진행한 결과입니다.

****

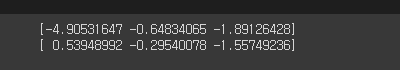
전처리 된 dataset의 속성정보와 임의의 data를 plot한 결과입니다.

tmp = myDenseLayer(3,5) 의 과정을 거치고, print(tmp.forward(np.random.randn(2,5,3)))

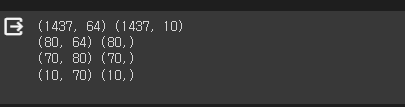
구축한forward function을 통과한 값을 print한 것입니다.

****

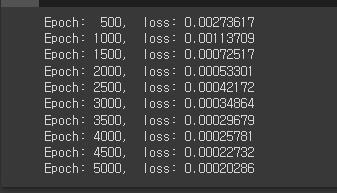
Backpropagation에 쓰이는 activation functions들을 정의한 후 random seed설정 후 그에 해당하는 data를 dJdz\_sigmoid와 dJdz\_softmax를 통과시킨 후의 값을 print한 것입니다.

****

NN model을 create한 후 matrix dimension을 check합니다.

****

Training simple NNmodel(3 layer model) 즉, 3개의 layer로 구성된 model을 구성한 뒤 epoch가 500부터 5000까지 500주기로 loss값을 출력하고 그 값을 이용하여서 weight를 조정한 뒤 다시 loss값을 출력하도록 합니다.

****

from sklearn.metrics import accuracy\_score 한 후 그에 해당하는 y의 prediction값과 y의 실제결과 값을 통해 정확도를 계산 후 출력합니다.

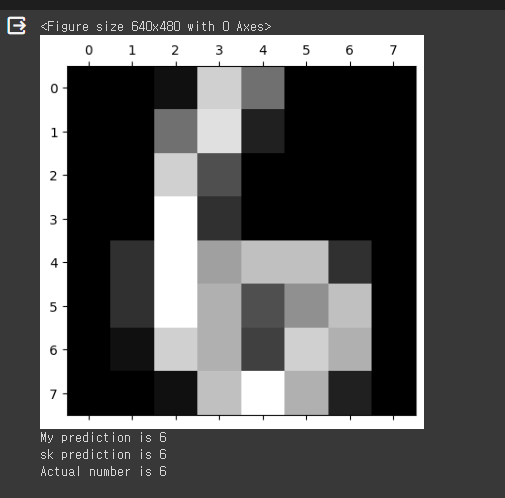
****

from sklearn.neural\_network import MLPClassifier

mlp = MLPClassifier(hidden\_layer\_sizes=(80, 70, ), activation='logistic', solver='sgd',alpha=0.01, learning\_rate\_init=0.01, max\_iter=1000)

****

Random sample을 통해 model을 test한 후 그 결과를 print out 합니다.

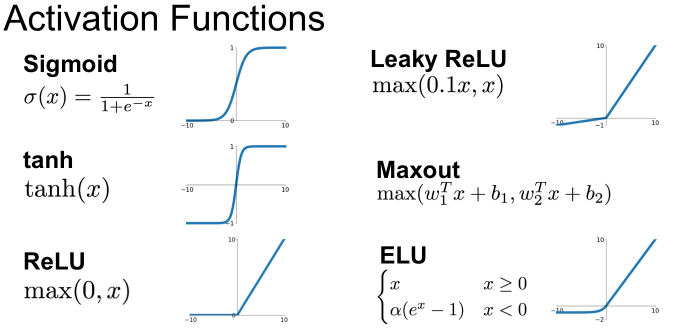
****

**<Discussion>**

처음 class myDenseLayer, model class를 정의하는 과정에서 expected outputs를 출력하기 위해서 식식 구성할 때 handout자료에 맞게 작성해야 했는데 DNN검색 후 일반적으로 많이 쓰이는 수식으로 하니까 출력이 안되고 matrix에 안 맞아 문제가 생겼었습니다. 또한 backward path에서 쓰이는 dJdz\_sigmoid와 dJdz\_softmax의 수식을 정의하는 과정에서도 간단하게 정리된 식을 두고 복잡하게 인자를 이용해서 써서 원했던 결과보다 값의 범위가 너무 넓게 나오는 등의 과정을 거쳤습니다. 또한 expected outputs은 아니지만 loss값이 1.xxxx…범위에서 시작하는 것이 아니라 0.000xxx…즉 loss값이 꽤 작을 때부터 시작 되는 것과 Random sample을 통해 model을 test한 후 그 결과를 print out한 결과에서 Figure size가 커진 것이 아마 Training Simple Neural Network Model의 과정에서 layer가 확장하는 대신 정확도가 올라간 것이 아닐까 하는 생각이 들었습니다. 또한 NN의 구조에 따라 다르게 적용되는 loss function과 앞서 언급한 training하는 dataset의 종류 혹은 그 특성 외의 것들에 따라 다르게 적용되는 activation functions의 종류도 추가로 알아본 결과 다음과 같습니다. 먼저 loss function의 종류와 개념을 간단히 정리하면 MSE, 예측한 값과 실제 값 사이의 평균제곱오차를 정의합니다.

RMSE는 MSE에 루트를 씌운 것으로 MSE와 기본적으로 동일합니다. 루트 때문에 MSE의 값의 왜곡을 줄여준다는 특징이 있습니다. Binary Crossentropy는 실제 레이블과 예측 레이블 간의 교차 엔트로피 손실을 계산하는 것으로 output의 출력이 2개의 label class가 있을 때 좋습니다.

Categorical Crossentropy는 label class가 2개 이상일 경우 사용하며 보통 softmax다음에 연계되어 나온다고 하여 softmax loss라고도 불립니다.



( [출처 cs231n] )

Sigmoid function의 특징과 단점은 다음과 같습니다. 특징은 출력 값을 0에서 1로 변경해준다는 것 입니다.(Squashes number to range [0, 1]) 또한, 가장 많이 사용되었던 활성화 함수라고 합니다.

단점으로는 Saturation(포화상태)과 Sigmoid outputs are not zero-centered 점입니다.

tanh의 특징과 단점은 다음과 같습니다. 특징은 출력 값을 -1에서 1로 압축시켜 준다는 것입니다.

zero-centerd 합니다. 이 특징 덕에 sigmoid가 가졌던 두 번째 문제점을 해결해줄 수 있습니다.

단점으로는 여전히 gradient가 죽는 구간이 있다는 것 입니다. (양수/음수 구간 모두 존재합니다)

ReLU의 특징과 단점은 다음과 같습니다. 특징은 양의 값에서는 Saturated 되지 않는다는 것과 계산 효율이 뛰어나다는 것입니다. Sigmoid,tanh보다 훨씬 빠릅니다. 단점으로는 zero-centerd가 아니라는 문제(non-zero centered)와 음수 영역에서 saturated 되는 문제가 다시 발생합니다. 그럼에도 불구하고 최근 가장 많이 쓰이는 모델입니다. Leaky ReLU의 특징은 다음과 같습니다. ReLU와 유사하지만 negative regime(음의 영역)에서 더 이상 0이 아닙니다. 또한 saturated 되지 않습니다 그럼에도 불구하고 계산이 효율적이며 빠릅니다 또한 더 이상 Dead ReLU현상이 없게 됩니다. 이번 과제를 통해서 Neural network의 구조와 그에 필요한 식들에 대해서도 그리고 데이터를 이용한 결과 확인까지의 절차를 거치면서 많이 배웠다고 생각합니다.