기초인공지능

Assignment4 보고서

20171646 박태윤

1. 직접 구현한 MLP model의 layer 구성에 대한 간단한 설명

MLP는 Multi-layer Perceptron의 약자로, single-layer Perceptron을 개선시킨 방식을 나타낸다. layer를 여러 개 순차적으로 두어 비선형적으로 분리되는 데이터에 대해 제대로 학습하지 못하는 기존의 single-layer Perceptron의 문제를 해결하는 딥 러닝 기술이다. 여러 층들 중에서 제일 위와 아래를 제외한 층들은 은닉층이라고 한다. 이번 AI04과제에서 MLP에 대한 코드는 다음과 같이 작성하였다.

class MLP(nn.Module):

    def \_\_init\_\_(self,input\_size,active\_func,output\_size):

        super(MLP,self).\_\_init\_\_()

        self.activation\_func=active\_func

        ######################## Write your Code Here ########################

        self.seq = nn.Sequential(

          nn.Linear(3072,1024),

          self.activation\_func,

          nn.Linear(1024,512),

          self.activation\_func,

          nn.Linear(512,256),

          self.activation\_func,

          nn.Linear(256,64),

          self.activation\_func,

          nn.Linear(64,16),

          self.activation\_func,

          nn.Linear(16,10),

          self.activation\_func

        )

        #######################################################################

    def forward(self,x):

        x = flatten(x)  # flatten layer: 위에서 정의한 flatten 함수를 먼저 작성해야 한다.

        x = self.seq(x) # 입력값이 하나인 것을 생각하고 network를 짜야한다.

        return x

flatten함수에 의해 x데이터의 사이즈가 3072로 평탄화가 되고, self.seq함수를 통해 layer의 진행을 나타낸다. self.seq에서 각 레이어 마다 output은 다음과 같은 순서로 구현하여 dimension을 줄여나갔다.

3072->1024->512->256->64->16->10

즉 해당 코드에서는 4개의 hidden layer와 input 그리고 output layer가 존재한다고 볼 수 있다.

또한 각 layer마다 activation\_func함수가 호출이 되는데, 이는 활성 함수로 이번 과제에서는 sigmod, relu, tanh, leakyrelu중 하나를 선택할 수 있다. 활성 함수란 입력 신호의 총합을 출력 신호로 변환해주는 함수이다.

2. 직접 구현한 CNN model의 layer구성에 대한 간단한 설명

CNN은 Convolutional Neural Network의 약자로, 이미지 관련 딥 러닝에서 사용되는 기술이다. 이에 대한 코드는 다음과 같이 작성하였다.

class CNNModel(nn.Module):

    def \_\_init\_\_(self,input\_channel,active\_func):

        super(CNNModel, self).\_\_init\_\_()

        self.activation\_func=active\_func

        ######################## Write your Code Here #########################

        self.fc1 = nn.Linear(5\*5\*24,120)

        self.fc2 = nn.Linear(120,60)

        self.fc3 = nn.Linear(60,10)

        self.layer1 = nn.Sequential(

            nn.Conv2d(3, 12, 3, stride = 1, padding = 0),

            self.activation\_func

        )

        self.layer2 = nn.Sequential(

            nn.Conv2d(12, 12, 3, stride = 1, padding = 0),

            self.activation\_func,

        )

        self.layer3 = nn.Sequential(

            nn.MaxPool2d(2),

            nn.Conv2d(12, 24, 3, stride = 1, padding = 0),

            self.activation\_func

        )

        self.layer4 = nn.Sequential(

            nn.Conv2d(24, 24, 3, stride = 1, padding = 0),

            self.activation\_func,

            nn.MaxPool2d(2)

        )

        #######################################################################

    def forward(self, output):

        ######################## Write your Code Here #########################

        output = self.layer1(output)

        output = self.layer2(output)

        output = self.layer3(output)

        output = self.layer4(output)

        length = len(output)

        output\_tmp = output.flatten()

        tmp\_len = len(output\_tmp)

        output = output\_tmp.reshape(length, tmp\_len // length)

        output = self.fc1(output)

        output = self.fc2(output)

        output = self.fc3(output)

        return output

        #######################################################################

총 4개의 layer가 존재한다.

layer1에서는 32x32x3 입력을 30x30x12로 변경을 해주어야 한다. 이를 위해 conv2d함수를 이용하여 input channel은 3, output channel은 12, kernel size는 3, stride와 padding은 각각 1과 0으로 설정하여 32x32x3 => 30x30x12 output을 생성하였다.

layer2에서는 30x30x12를 28x28x12로 convolution해준다. 역시 conv2d함수를 이용하여 input channel은 12, output channel도 12, kernel size는 30에서 28로 2를 줄여야 하기 때문에 3으로, stride와 padding역시 각각 1과 0으로 설정하여 28x28x12의 output을 생성하였다.

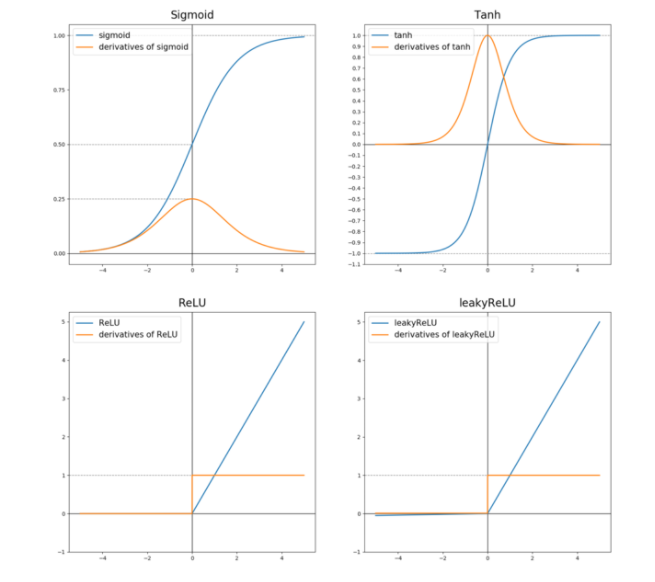
layer3에서는 기존에 layer2에서 convolution이 일어난 데이터에 pooling작업을 가해 만들어진 14x14x12데이터를 가지고 12x12x24데이터를 생성해야 한다. 따라서 이를 위해 nn.MaxPool2d(2)를 호출하여 28x28x12 => 14x14x12로 pooling해주었고, 이를 통해 conv2d함수에서 input channel을 12, output channel을 24, kernel size를 3, stride와 padding을 각각 1과 0으로 설정하여 12x12x24로 convolution하였다.

layer4에서는 12x12x24를 가지고 10x10x24로 convolution한 뒤 5x5x24로 pooling을 해주면 된다. 따라서 conv2d함수에서 input output channel을 24로, kernel size를 3으로, stride와 padding을 각각 1과 0으로 설정하여 10x10x24로 convolution한 뒤 nn.MaxPool2d(2)를 통해 5x5x24로 pooling해주었다.

3. 선택한 Activation function에 대한 설명

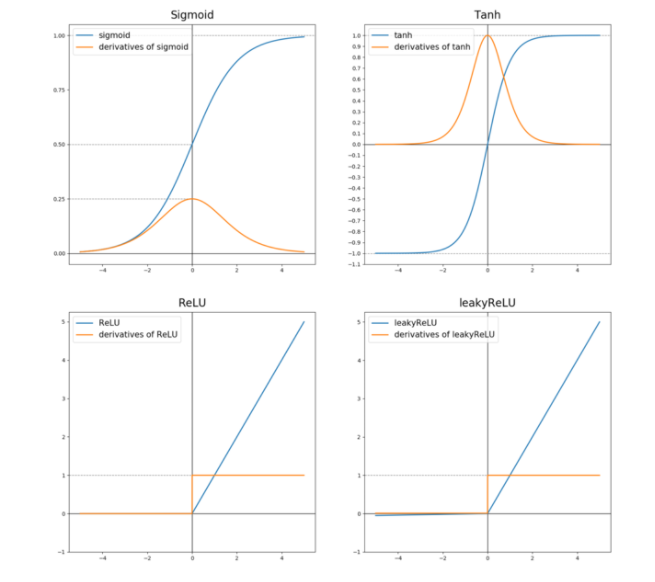
위 과제에서 sigmod, relu, tanh, leakyrelu총 4개의 활성 함수 중 하나를 선택해서 과제를 진행할 수 있는데, leakyrelu를 선택하였다. 이를 선택한 이유는 다음과 같다.

-sigmoid



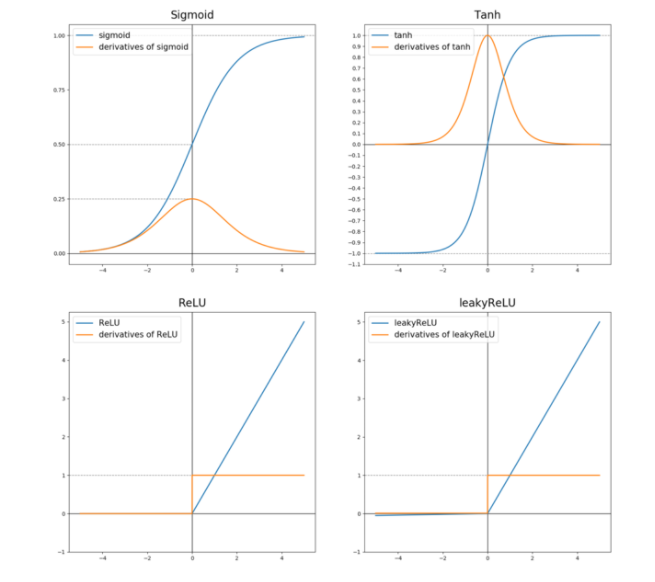
output을 0에서 1사이로 만들어주는 sigmoid 활성화 함수인데, 그래프를 보면 input의 절댓값이 어느정도 커지는 순간 기울기가 매우 작아져 여러 layer를 겹겹이 쌓았을 때 역전파를 해갈수록 대부분 노드에서 기울기가 0이 되어 gradient가 결국 완전히 사라지는 vanishing gradient문제가 발생할 수 있다.

- tanh



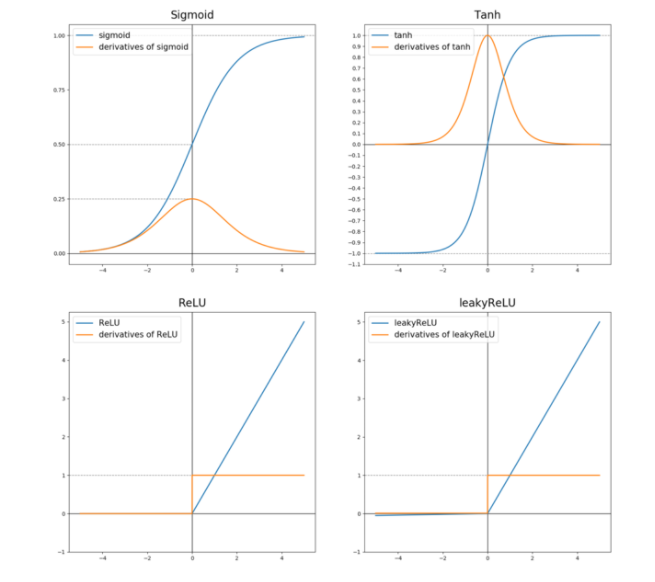
tanh역시 sigmoid와 마찬가지로 입력의 절댓값이 어느정도 커지는 순간 기울기가 지나치게 작아지기 때문에 vanishing gradient문제가 존재한다.

- relu



vanishing gradient문제가 없는 relu활성 함수이다. 입력 값이 0보다 작은 음수일 경우 기울기가 0인데, 대부분 hidden layer에서 노드의 z값이 0보다 크다는 것을 감안하여 이와 같은 활성 함수를 사용할 수 있다.

-leakyrelu



relu와 달리 음수일 때 기울기가 0이 아니라 0.01z를 갖게 되는 활성 함수이다. 이는 relu에서 0보다 작은 입력 값에 대해 처리를 해준다는 장점이 존재하여 더 높은 학습 효과를 기대할 수 있기 때문에 이를 선택하여 과제를 진행하였다.

4. 실행 결과



