과목 명: 기초인공지능

담당 교수 명: 양 지 훈

<<AI_assignment04>>

서강대학교 컴퓨터학과

[학번] 20171700

[이름] 최재원

목 차

1. MLP model

단일층 모델을 이용하여 classification 과 같은 문제를 해결하기 위해 학습을 진행하면, decision boundary 가 linear 함을 벗어나지 못하는 문제가 있다. 따라서 이를 극복하기 위해 Multi Level Perception 모델에서는 layer 수를 늘림으로써, 즉 Input layer 와 Output layer 사이에 다중의 hidden layer 를 둠으로써, decision boundary 를 선형에서 비선형, 혹은 평면에서 초평면의 형태로 복잡하게 표현할 수 있도록 한다. 각 층의 unit 수를 늘림으로써 더 높은 차원으로 문제에 접근 가능해 지는 것이다. 아래는 이번 과제에서 구성한 MLP 모델에대한 layer 에 관한 코드이다.

```
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 ########################
        # (input size, hidden size)
        self.layer1 = nn.Linear(in_features=1024*3,out_features=1024,bias=True)
        self.layer2 = nn.Linear(in_features=1024,out_features=1024,bias=True)
        self.layer3 = nn.Linear(in_features=1024,out_features=1024,bias=True)
        self.layer4 = nn.Linear(in_features=1024,out_features=1024,bias=True)
        self.layer5 = nn.Linear(in_features=1024,out_features=1024,bias=True)
        self.layer6 = nn.Linear(in_features=1024,out_features=1024,bias=True)
        self.layer7 = nn.Linear(in_features=1024,out_features=1024,bias=True)
        self.layer8 = nn.Linear(in_features=1024,out_features=512,bias=True)

        self.seq = nn.Sequential(
            self.layer1,
            self.activation_func,
            self.layer2,
            self.activation_func,
            self.layer3,
            self.activation_func,
            self.layer4,
            self.activation_func,
            self.layer5,
            self.activation_func,
            self.layer6,
            self.activation_func,
            self.layer7,
            self.activation_func,
            self.layer8
        )

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

    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
```

위와 같이 총 8 개의 layer 로 구성하였고, layer1 가 Input layer 를, layer8 이 output layer 를 의미하며 그 사이의 layer 들은 hidden layer 를 의미한다. 다층 신경망에서는 layer 수가 많아질수록 계산이 복잡해지는데, 이를 해결하기 위해서 Back propagation

algorithm 을 사용하기도 한다. Output layer(layer8)에 가까운 순으로 bias 와 weight 에 대한 계산을 진행하여, 그 전 layer 로 전달하는 것이다. 이를 위해서 activation function 으로 비선형 활성화 함수를 사용한다. 비선현 활성화 함수를 사용하면 심층 신경망을 통해 더 많은 핵심 정보를 얻을 수 있고, back propagation 또한 가능하다. 여러 비선형 활성화 함수중에 어떤 함수를 사용하는지는 뒷부분에서 설명하도록 한다. Ipnut layer 의 뉴런 수는 1024*3 인 것을 볼 수 있는데, 이는 CIFAR10 의 input shape 을 고려한 것이다. 그리고 hidden layer 의 뉴런수는 1024, output layter 의 뉴런수 는 512 로 정하였다. 앞에서 설명한 이유들로, 적절한 연산 효율성을 위해서 layer8 까지로 설정하였고, flatten 함수에서는 reshape 을 통해 flattening 을 구현해 주었다.

2. CNN model

```
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.L1 = nn.Sequential(
            nn.Conv2d(in_channels=3,out_channels=12,kernel_size=(5,5),stride=1,padding =1),
            self.activation_func,
            nn.Conv2d(in_channels=12,out_channels=12,kernel_size=(5,5),stride=1,padding =1),
            self.activation_func,
            nn.MaxPool2d(kernel_size=2,stride = 2)
        )
        self.L2 = nn.Sequential(
            nn.Conv2d(in_channels=12,out_channels=24,kernel_size=(5,5),stride=1,padding =1),
            self.activation_func,
            nn.Conv2d(in_channels=24,out_channels=24,kernel_size=(5,5),stride=1,padding =1),
            self.activation_func,
            nn.MaxPool2d(kernel_size=2,stride = 2)
        )


        self.F1 = nn.Sequential(
            nn.Linear(in_features=600,out_features=10,bias=True) ,
            self.activation_func
        )
        self.F2 = nn.Sequential(
            nn.Linear(in_features=10,out_features=10,bias=True) ,
            self.activation_func
        )
        self.F3 = nn.Sequential(
            nn.Linear(in_features=10,out_features=10,bias=True) ,
            self.activation_func
        )
        ######################################################################

    def forward(self, output):
        ######################## Write your Code Here ########################
        x = self.L1(output)
        x = self.L2(x)

        x = x.view(x.size(0),-1)
        x = self.F1(x)
        x = self.F2(x)
        x = self.F3(x)
        return x
```

Convolutional Neural Network 에서는 합성곱(convolution) 연산을 사용하여 학습을 진행하는 ANN 이다. 각 layer 의 입출력 형상을 유지하며, 이미지의 공간 정보를 유지하면서 인접 이미지와의 특징을 효과적으로 인식하는 등의 특징이 있다. 다만 FC layer 는 1 차원 데이터만 입력 받을 수 있기 때문에, 3 차원 데이터를 flattening 해서 입력해야 한다. 이 때 공간적으로 정보가 소실되는 문제가 발생한다. 반면에 Conv

layer 는 형상을 유지할 수 있다. 입출력 모두 3 차원 데이터로 처리하기 때문이다. 과제에서 주어진 INPUT 은 32*32*3, 즉 32*32 Pixel 이 3 channel 로 구성되어 들어오게 되며, Conv layer 를 3 개 요구하고 있으며, FC layer 는 2 개에서 4 개 사이로 요구하고 있다. Convd 에 대해서는 Conv2d 를 사용하였고, 이 때 각 padding, kernel size, stride 를 사용하는데 padding 은 출력 데이터가 줄어드는 것을 방지하기 위해 보통 0(zero)으로 값을 채워넣는 것을 의미하며, padding = 1 은 데이터 외각에 1pixel 씩을 추가하는 것을 의미한다. 이를 구하기 위해 사용한 식은 아래와 같다.

- Input: $(N, C_{in}, H_{in}, W_{in})$
- Output: $(N, C_{out}, H_{out}, W_{out})$ where

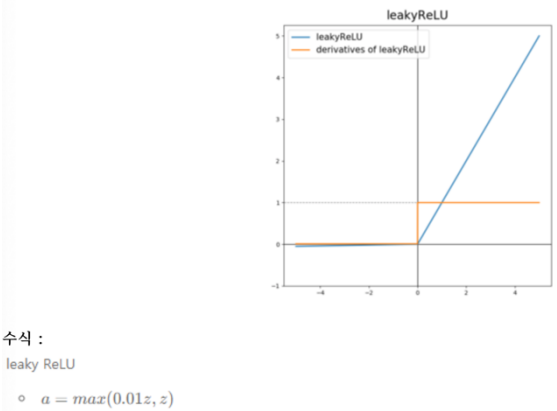
$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

Conv2d 를 사용해 convolution layer 에서 fileter 를 이용해 이미지 feature 를 계산하고, activation function 을 사용해 정리해준 다음, 계산 결과(feature maps)를 pooling, 즉 sub sampling 을 해준다. 이 때 Conv2d 에 대응되는 Maxpool2d 를 사용하였고 그 kernel size 는 2 이며, stride 도 2 이다. maxpooling 은 정해진 크기 안에서 가장 큰 값만 뽑아낸다. Pooling 의 목적은 input size 를 줄이고(down sampling), overfitting 을 조절하며, 특징을 더 잘 뽑아내는 것에 있다. Forward 함수에서 out.view 는 batch size 를 기준으로 한 row 를 만들며, 나머지 값을 그 열에다 입력하도록 한다.

3. Activation function

앞에서 설명한대로 activation function 은 선형, 비선형 등이 있지만 과제에서 주어진 것은 비선형 황성화 함수들이며, 결국 선형 항목과 비선형 항목들을 서로 교차시키는 역할을 한다. Sigmod 함수보다는 ReLU 함수가 더 간단하고, ReLU 는 대부분의 Input 값에 대해서 기울기가 0 이 아니기 때문에 학습이 빨리 된다. 학습을 느리게하는 원인이 gradient 가 0 이 되는 것인데, 이를 대부분의 경우에서 막아주기 때문에, sigmoid, Tanh 같은 함수보다 학습이 빠르다. 내가 이번 과제에서 선택한 활성화 함수는 leaky ReLU 함수인데, ReLU 와 유일한 차이점으로는 max(0, z) 대신 max(0.01z, z)를 사용한다는 점이다. 즉, input 값인 z 가 음수일 경우 기울기가 0 이 아닌 0.01 을 갖게된다. 앞에 언급한대로 기울기가 0 인 부분에 대해서 학습이 느려지기



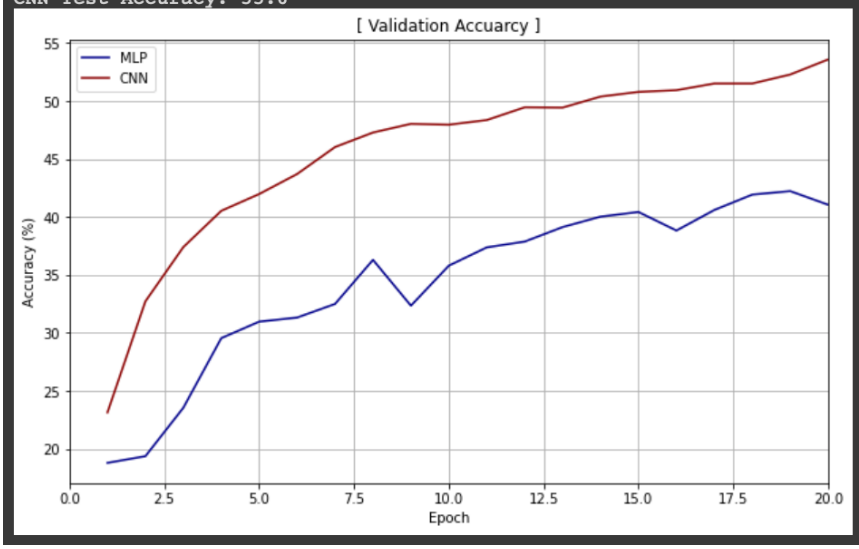
수식 :

leaky ReLU

- $a = max(0.01z, z)$

때문에, input 값이 음수일 때 ReLU 보다 leaky ReLU 가 학습이 더 잘되기 때문에 해당 활성화 함수를 선택하였다.

```
---------- MLP Test Result ----------
MLP Test Accuracy: 43.1
============= CNN Train Start ================
1th epoch loss:2.188958 Train Acc:17.94375
2th epoch loss:1.948888 Train Acc:27.1
3th epoch loss:1.763789 Train Acc:35.34375
4th epoch loss:1.667904 Train Acc:38.9375
5th epoch loss:1.607374 Train Acc:40.83125
6th epoch loss:1.538941 Train Acc:43.31875
7th epoch loss:1.489200 Train Acc:45.5625
8th epoch loss:1.441550 Train Acc:47.6
9th epoch loss:1.401035 Train Acc:49.10625
10th epoch loss:1.363158 Train Acc:50.71875
11th epoch loss:1.333787 Train Acc:51.96875
12th epoch loss:1.304019 Train Acc:53.01875
13th epoch loss:1.277303 Train Acc:53.95
14th epoch loss:1.247649 Train Acc:55.325
15th epoch loss:1.219340 Train Acc:56.525
16th epoch loss:1.197189 Train Acc:57.1375
17th epoch loss:1.174366 Train Acc:58.025
18th epoch loss:1.154286 Train Acc:58.825
19th epoch loss:1.136143 Train Acc:59.49375
20th epoch loss:1.111066 Train Acc:60.5
---------- CNN Test Result ----------
CNN Test Accuracy: 55.6
```



위는 MLP, CNN 에 대해서 activation function 을 leaky ReLU 를 활용하였을때의 test Accuracy 및 validation accuracy graph 화면이다. MLP 에서는 hidden layer 의 neuron 수를 input layer 에서보다 더 증가시키거나 그 수를 변칙적으로 주었을 때 accuracy 가 더 낮아짐을 확인하였고, 여러 실험을 통해 위에 언급한 Layer 설정을 갖추게 되었다. MLP 의 accuracy 는 약 43%로 측정되었고, layer 수가 문제에서 주어진 것보다 더 많아질 경우 overfitting 등의 이유로 accuracy 가 낮아질 것이라는 추측을 할 수 있다. CNN 에서는 4 개의 convolution layer, 3 개의 Fully connected layer 를 설정하여 약 55%의 accuracy 가 측정되었다.