기초 인공지능 Assignment03 – From Sentence to Logic 보고서

2021.11.06 11:00 PM

20181202 김수미

1. 직접 구현한 MLP model의 layer 구성에 대한 간단한 설명

```
def flatten(x):
    data = x.shape[0]
    ################### Write Your Code Here ###################
    flatten_x = torch.flatten(x, start_dim=1, end_dim=-1)
    ############################################################
    return flatten_x
```

```
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(input_size, 600),
                                 self.activation_func,
                                 nn.Linear(600, 300),
                                 nn.Linear(300, 200),
                                 self.activation_func,
                                 nn.Linear(200, 100),
                                 nn.Linear(100, 10))
        ########################################################

    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
```

총 5개의 Linear layer와 1개의 Flatten layer 배치하고 activation function을 두 번 사용하여 Multi-Layer Perceptron을 구성하였다.

Flatten layer란 Convolution Neural Network 의 데이터 타입을 Fully Connected Neural Network의 형태로 변경하는 layer로, torch 라이브러리에서 제공하는 torch.flatten 함수를 사용하여 구현할 수 있다. torch.flatten 함수는 인자로 flatten 의 대상이 되는 데이터와 start_dim, end_dim 값을 받는데, 데이터의 구성이 예를들어 [a,b,c,d] 인 경우 start_dim, end_dim 을 각각 1, 3으로 설정하면 1부터 3의 인덱스에 해당하는 b,c,d가 flatten 되어 [a, 〈flatten〉] 식의 결과가 만들어지게 된다.

nn.Linear 함수는 float타입의 input 데이터에 대해 y=wx+b 형태의 선형 변환을 수행하는 메소드이다. y=wx+b 형태이기 때문에 weight와 bias 값의 업데이트를 통해서 학습을 수행한다고 이야기할 수 있다.

Activation Function의 역할은 입력된 데이터의 weighted sum을 출력 신호로 변환하는 함수이다.

Network에서 이전 레이어에 대한 weighted sum의 크기에 따라 활성 여부가 결정된다. Activation Function을 사용함으로써 비로소 여러개의 레이어를 이용해 네트워크를 구성하는 것에 대한 이득을 취할 수 있다.

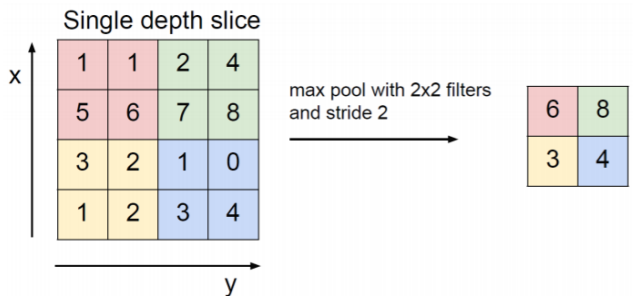
2. 직접 구현한 CNN model의 layer 구성에 대한 간단한 설명

```
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.net = nn.Sequential(
                    nn.Conv2d(input_channel, 12, kernel_size=5, stride=1, padding=1), self.activation_func,
                    nn.Conv2d(12, 12, kernel_size=5, stride=1, padding=1), self.activation_func,
                    nn.MaxPool2d(kernel_size=1, stride=2),

                    nn.Conv2d(12, 24, kernel_size=5, padding=1), self.activation_func,
                    nn.Conv2d(24, 24, kernel_size=5, padding=1), self.activation_func,
                    nn.MaxPool2d(kernel_size=1, stride=2), nn.Flatten(),

                    nn.Linear(600, 300), self.activation_func, nn.Dropout(p=0.5),
                    nn.Linear(300, 100), nn.Dropout(p=0.5),
                    nn.Linear(100, 10))
        ######################################################################

    def forward(self, output):
        ######################## Write your Code Here ########################
        output = self.net(output)
        return output
        ######################################################################
```

총 4개 Conv2d 레이어 외 2번의 max pooling, 5번의 activation function, 1개의 flatten layer, 총 3개의 Linear layer 등을 이용하여 CNN 모델을 구성하였다. 정확도를 향상시키기 위해 Dropout 기법도 추가해 주었다.

먼저 nn.Conv2d 함수는 pytorch에서 사용할 수 있는 convolution layer이다. 여기서 convolution layer란 stride 값 만큼씩 이동하면서 겹쳐지는 부분의 각 원소의 값을 곱하여서 모두 더한 값을 출력하는 층이다.

다음으로 Pooling 이란 간단히 말해서 특징을 뽑아내는 작업이라고 할 수 있다.



이 때 Max Pooling 이란 정해진 크기 안에서 가장 큰 값을 뽑아내는 작업을 의미한다. Pooling 작업은 input size 와 overfitting 을 줄이는 작업이기에 정확도를 향상시키는 것에 도움이 된다.

Dropout 이란 Neural Network 의 뉴런을 부분적으로 생략하여 모델의 overfitting 을 줄이기 위한 방법 중 하나이다. 이 역시 모델의 정확도 향상에 많은 도움을 준다.

즉 전체적으로 과제에서 제시한 구조인 INPUT(32X32X3) → CONV1(30X30X12) → CONV2(28X28X12) → POOL1(14X14X12) → CONV3(12X12X24) → CONV4(10X10X24) → POOL2(5X5X24) → FC(3 개) 를 따라 Model 을 작성하였다.

3. 선택한 Activation function에 대한 설명 및 선택 이유

```
# hyperparameter
EPOCHS = 20
LEARNING_RATE = 1e-3

## 배치 사이즈는 사양에 맞게 변경이 가능하다.
###################################
BATCH_SIZE = 128
###################################

train_loader = DataLoader(Custom_Dataset(X_train,Y_train), batch_size = BATCH_SIZE)
val_loader = DataLoader(Custom_Dataset(X_valid,Y_valid), batch_size = BATCH_SIZE)
test_loader = DataLoader(Custom_Dataset(X_test,Y_test), batch_size = BATCH_SIZE)

########## Write Your Code Here #########
active_function = choice_activation("relu")
#########################################
criterion = nn.CrossEntropyLoss()
```

선택한 activation function은 ReLU 이다. 이 함수는 0보다 작은 값이 나온 경우 0을 반환하고, 0보다 큰 값이 나온 경우 그 값을 그대로 반환하는 함수이다. 연산이 간단하기 때문에 학습 속도 면에서 매우 유리하며, hidden layers에 ReLU를 적용해주면 model의 정확도를 향상시키는데 도움이 된다. ReLU 함수는 가장 많이 사용되는 활성화 함수 중 하나이며 Sigmoid 함수와 tanh 함수가 가지는 Gradient Vanishing 문제를 해결해주는 함수이기도 하다. 이러한 이유 때문에 과제에서 제시된 4개의 함수 중에서 ReLU 함수를 선택하게 되었다.

4. MLP Test Acuuracy와 CNN Test Accuracy 캡처 화면

```
---------- MLP Test Result ----------------
MLP Test Accuracy: 45.325

---------- CNN Test Result ----------------
CNN Test Accuracy: 53.075
```

5. Validation Accuracy Graph 캡처 화면