손글씨 분류하기

손글씨 분류하기

손글씨 분류하기

- 손글씨 이미지를 입력받아 어떤 숫자인지 예측하기
- 알고리즘: MLP
- 데이터셋 파일명 : MNIST
  - 출처: http://yann.lecun.com/exdb/mnist

- 0부터 9까지 10개의 숫에 대한 손글씨 이미지를 모아놓은 데이터
- 분류 유형

손글씨 분류하기

- 활성화함수
  - 시그모이드 함수 : 0과1 상이로 제한하는 함수
    - nn.Sigmoid()
  - 렐루 함수 : 0보다 작은 값은 0으로, 0보다 크거나 같은 값은 입력값을 그대로 출력
    - ReLU 함수는 정류 선형 유닛(영어: Rectified Linear Unit 렉티파이드 리니어 유닛[*])에 대한 함수이다. ReLU는 입력값이 0보다 작으면 0으로 출력, 0보다 크면 입력값 그대로 출력하는 유닛 (출처:위키백과)
    - nn.ReLU()
  - 소프트맥스(벡터 함수므로 그림으로 나타낼 수 없음) : k개의 숫자를 입력받아 k개의 요소를 갖는 활률 분포 변환하는 함수. 여러 클래스에 속할 확률을 타나내므로 다중분류에 사용.
    - nn.Softmax()

손글씨 분류하기

- nn.ReLU(): ReLU(Rectified Linear Unit) 활성화 함수(activation function)를 적용합니다. 이 함수는 입력값이 0보다 작을 경우 0을 출력하고, 0보다 큰 경우 입력값을 그대로 출력합니다.
- nn.Sequential: 여러 개의 layer를 차례로 쌓아서 신경망 모델을 만들기 위한 함수로, Sequential 안에 들어가는 layer는 순서대로 연결됨.
- nn.Linear(784,64)
  - MNIST 숫자 분류 문제가 있습니다. MNIST 데이터셋은 28x28 픽셀 크기의 흑백 이미지로 이루어져 있습니다. 각 이미지는 784(=28x28)개의 특성을 가진다.
- Adam(optimizer)은 파이토치(PyTorch)에서 제공하는 최적화 알고리즘 중 하나
  - Adam은 딥 러닝에서 가장 많이 사용되는 optimizer 중 하나입니다.

손글씨 분류하기

- optim=Adam(model.parameters(), lr=lr)
- 옵티마이저(Optimizer)
- 최적화는 각 학습 단계에서 모델의 오류를 줄이기 위해 모델 매개변수를 조정하는 과정
- PyTorch에는 ADAM이나 RMSProp과 같은 다른 종류의 모델과 데이터에서 더 잘 동작하는 다양한 옵티마이저가 있습니다.

손글씨 분류하기

- torch.reshape(data, (-1, 784))는 입력 데이터(data)를 (-1, 784) 크기의 2차원 텐서로 변환하는 코드입니다. 이 코드에서 -1은 입력 데이터의 첫 번째 차원은 자동으로 조정되고, 두 번째 차원은 784로 고정됩니다.

- -1 : 개수를 상관하지 않음.
- Batch 32 이므로 32*28,28 = 32*784 = (-1,784) = (32,784)

  모든 배치크기,784 모든 픽셀이 다음과 같이 묶여서 데이터가 변경된다. 이것을 우리 모델의 입력으로 넣는다.
- MLP
- 확률 분포의 차이
- 예측분포, 실제의 데이터의 확률분포의 차이를 나타낸다.

용어 정리

- 파이토치(PyTorch) MLP(Multi-Layer Perceptron)은 인공신경망의 가장 기본적인 형태로, 여러 개의 선형 계층(linear layer)과 활성화 함수(activation function)를 쌓아올린 모델
- MLP는 일반적으로 입력층(input layer), 은닉층(hidden layer), 출력층(output layer)으로 구성되며, 각 층은 서로 다른 수의 노드를 가질 수 있다
- MLP는 가중치(weight)와 편향(bias)을 사용하여 연산을 수행하고, 입력층과 은닉층의 각 노드는 (linear layer)로 구성되며, 출력층의 노드는 (activation function)를 통해 최종 출력값을 생성.
- MLP에서는 일반적으로 ReLU(Rectified Linear Unit) 활성화 함수를 사용한다.

용어 정리

- 파이토치에서는 MLP를 구현할 때 nn.Module 클래스를 상속받아 모델을 정의하고, nn.Linear 클래스와 nn.ReLU 클래스를 사용하여 각각의 선형 계층과 활성화 함수를 구성합니다. 또한 optimizer(최적화 알고리즘)과 loss function(손실 함수)을 설정하여 학습을 수행한다.

용어정리

- 하이퍼파라미터(Hyperparameter)
- 하이퍼파라미터(Hyperparameter)는 모델 최적화 과정을 제어할 수 있는 조절 가능한 매개변수입니다. 서로 다른 하이퍼파라미터 값은 모델 학습과 수렴율(convergence rate)에 영향을 미칠 수 있습니다. (하이퍼파라미터 튜닝(tuning)에 대해 더 알아보기)

- 학습 시에는 다음과 같은 하이퍼파라미터를 정의합니다:
- 에폭(epoch) 수 - 데이터셋을 반복하는 횟수

- 배치 크기(batch size) - 매개변수가 갱신되기 전 신경망을 통해 전파된 데이터 샘플의 수

- 학습률(learning rate) - 각 배치/에폭에서 모델의 매개변수를 조절하는 비율. 값이 작을수록 학습 속도가 느려지고, 값이 크면 학습 중 예측할 수 없는 동작이 발생할 수 있습니다.

코드

```
import matplotlib.pyplot as plt

from torchvision.datasets.mnist import MNIST
from torchvision.transforms import ToTensor

training_data=MNIST(root='./',train=True, download=True, transform=ToTensor())
test_data=MNIST(root='./',train=False, download=True, transform=ToTensor())

print(len(training_data))
print(len(test_data))

for i in range(9):
  plt.subplot(3,3,i+1)
  plt.imshow(training_data.data[i])
plt.show()
```

코드

```
from torch.utils.data.dataloader import DataLoader

train_loader=DataLoader(training_data,batch_size=32,shuffle=True)
test_loader=DataLoader(test_data,batch_size=32,shuffle=False)
plt.show()
```

코드

```
import torch
import torch.nn as nn
from torch.optim.adam import Adam

device ='cuda' if torch.cuda.is_available() else 'cpu'

model=nn.Sequential(
    nn.Linear(784,64),  #28*28*1 =784 64출력으로 내보내기
    nn.ReLU(),
    nn.Linear(64,64),
    nn.ReLU(),
    nn.Linear(64,10)       # 10개 출력 내보내기
)
model.to(device)
```

코드

```
lr=1e-3
optim=Adam(model.parameters(),lr=lr)

for epoch in range(20):
  for data, label in train_loader:
    optim.zero_grad()
    data=torch.reshape(data,(-1,784)).to(device)  #입력 데이터 -
1은 개수를 상관하지 않음.
    preds=model(data)     # 32*28*28=32*784 = (-1,784) = 32,784

    loss=nn.CrossEntropyLoss()(preds,label.to(device))
    loss.backward()
    optim.step()
  print(f"epoch{epoch+1} loss:{loss.item()}")

torch.save(model.state_dict(), "MNIST.pth")  #딕셔너리 state_dict()
```

코드

```
model.load_state_dict(torch.load('MNIST.pth',map_location=device))

num_corr = 0

with torch.no_grad():
  for data, label in test_loader:
    data=torch.reshape(data,(-1,784)).to(device)

    output=model(data.to(device))
    preds=output.data.max(1)[1]  # 예측값 구하기

    corr=preds.eq(label.to(device).data).sum().item()
    num_corr+=corr

  print(f"Accuracy:{num_corr/len(test_data)}")
```

감사합니다