C1. 파이토치 기본

torch 모듈 → 변수저장 텐서 → x= torch.tensor(3.5)

⇒ tensor(3.5000)

기울기 $\frac{dx}{dy}$ ⇒ y.backward ≒ x.grad

기본 계산식 다루기

C2. 파이토치로 만드는 신경망

MNIST 이미지 데이터셋

csv 다루기 → pandas 라이브러리 ⇒ DF

df.head() 상위 5개

df.info() 구성확인

시각화 → matplolib라이브러리 (matplolib.pyplot as plt)

데이터 할당 row = number, data= df.iloc[row] //관심대상

label label = data[0]

이미지 img= data[1:].values.reshape(28.28)

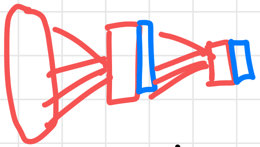
plt.title("label=" +str(label))

//비트맵 plt.imshow(img, interpolation='none', camp='Blues')

plt.show() //픽셀 스무 //컬러팔레트

간단한 신경망

28x28 이미지 → 입력레이어 784개
출력레이어 0~9개



은닉레이어 / 활성화함수
// 로지스틱회귀

신경망 → torch.nn as nn

분류기 nn.Module로부터 상속 받음

```
class Classifier(nn.Module):

    def __init__(self): // 초기화, 생성자

        super().__init__()
        self.model=nn.Sequential( // 신경망 레이어 정의
            nn.Linear(784,200), // 784-200 완전연결매핑
            nn.sigmoid(), // S모양 로지스틱 활성화 함수
            nn.Linear(200, 10), // 200→10연결(은닉, 가중치)
            nn.sigmoid() ) // 최종출력
```

MSE : 평균제곱오차 ⇒ torch.nn.MSELoss() // 손실함수

SGD : 확률적경사하강법 // 옵티마이저

```
self.optimizer.zero-grad()
loss.backward()
self.optimiser.step()
```

$\Big)$ // 기울기초기화, 역전파 실행, 가중치 갱신

훈련 시각화하기

train() 함수 안에

```
self.counter += 1

if(self.count%10==0):
    self.progress.append(loss.item()) //0~9, 텐서에서 값을 꺼내오는 함수
    pass
```

```
if(self.counter % 10000 ==0):    //속도확인
    print("counter=", self.count)
    pass
```

```
def plot_progress(self):
                                                //손실값저장
    df=pandas.DataFrame(self.progress, columns=['loss']) //판다스 DF로 변환

    df.plot(ylim=(0,1.0), figsize=(16,8), alpha=0.1, marker='.',
grid=True, yticks=(0,0.25,0.5)) //plot 함수
    pass
```

MNIST 데이터셋 클래스

파이토치: 데이터 셔플, 병렬, 배치

→ from torch.utils.data import Dataset

```
class MnistDataset(Dataset):
    def __init__(self, csv_file):
        self.data_df = pandas.read_csv(csv_file, header=None)
        pass
    def __len__(self): //데이터셋 길이 반환
        return len(self.data_df)
    def __getitem__(self, index): //데이터셋의 n번째 아이템 반환
        label = self.data_df.iloc[index 0]          원핫인코딩
        target = torch.zeros((10))                  0→[1000...00]
        target[label] = 1.0                         4→[0001...00]
        image_values = torch.FloatTensor(self.data_df.iloc[index, 1:].values)
                        /255.0 // 0-255 → 0-1로 정규화
        return label, image_values, target //레이블, 이미지 데이터 텐서,
                                             목표텐서 반환
    def plot_image(self, index):
        img = self.data_df.iloc[index, 1:].values.reshape(28, 28)
        plt.title("label=" + str(self.data_df.iloc[index, 0]))
        plt.imshow(img, interpolation='none', camp='Blues')
        pass //특정 이미지를 골라 차트를 그려보는 메소드
    pass

mnist_dataset = MnistDataset('mount/My Drive/Colab .../mnist_train.csv')
mnist_dataset.plot_image(a) // 10번째 데이터
```

분류기 훈련시키기

```
%%time //셀실행시간
    C=Classifier //신경망 생성
    epochs=3
    for i in range(epochs):
        print('training epoch',i+1,"of",epochs)
        for label,image_data_tensor,target_tensor in mnist_dataset:
            C.train (image_data_tensor,target_tensor) //훈련 진행
            pass
        pass
        //10000번의 호출마다 train() 메서드 진행 출력
    C.plot_progress() // 분류기 오차 출력
```

신경망에 쿼리하기

```
                                          //인덱스
image_data=mnist_test_dataset[record][1]
output=C.forward(image_data) // 훈련된 신경망으로부터 쿼리
pandas.DataFrame(output.detach().numpy()).plot(kind='bar',
//텐서출력 , 막대도표                              legend=False, ylim=(0,1))
  (신경망이 예상한 정답이 제일 큰값)
```

분류기의 성능 //test data 분류 결과 옳을때마다 score+1

```
score=0, items=0
for label,image_data_tensor,target_tensor in mnist_test_dataset:
    answer=C.forward(image_data_tensor).detach().numpy()
    if(answer.argmax()==label): //텐서의 어떤값이 제일 큰지
        score +=1
        pass
    items +=1
    pass
print(score,items,score/items) // score
```