KECE470: Pattern Recognition

School of Electrical Engineering, KOREA UNIVERSITY

(Homework #3) Artificial Neural Networks Report

2016170994 김다민

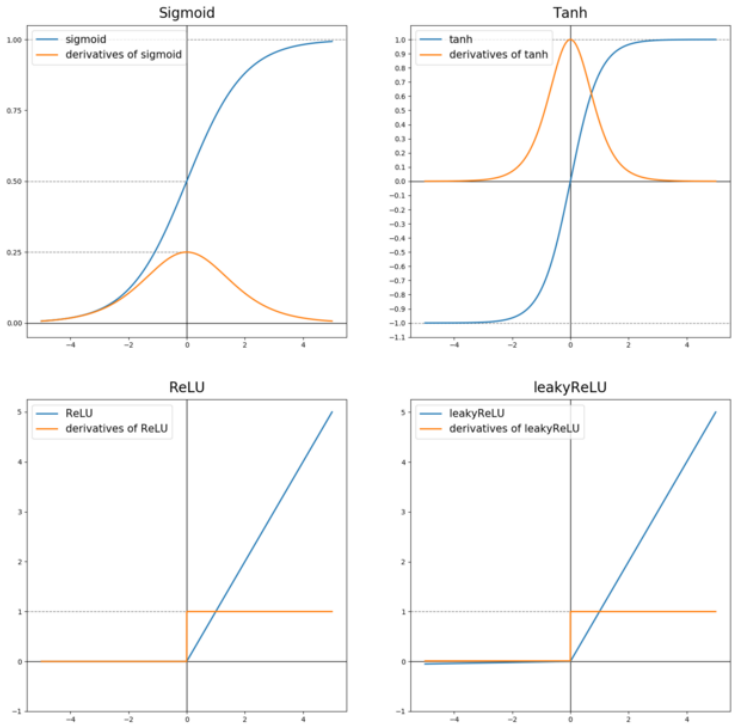
pytorch 라이브러리를 이용해서 구현해보았다.

1. **(Download MNIST dataset)** The data has been divided into several sets for training and test. You will randomly take 10% of the training set as validation set.

Torchvision 이라는 라이브러리를 이용해 pytorch 를 이용하도록 처리된 Mnist 데이터셋을 받았다. 원본과 동일하게 trainset 은 6 만개 testset 은 1 만개이며 6 만개의 train 데이터중 10%인 6 천개를 나누어 validationset 으로 이용했다.

2. **(Explain Activation Function)** Describe the role of the activation function, and give examples such as sigmoid and ReLU.

Activation function 은 network 에 nonlinear 한 성질을 추가해준다. 즉 이전 층에서 특정 임계값 주변에서 값차이를 크게 만들어 해당 임계값 기준으로 노드를 활성화 할 것인지 비활성화 할 것인지(다음 노드 값에 기여를 많이 할 것 인지 적게 할 것 인지) 비선형적인 경계역할을 할 수 있다.



Sigmoid

s자형 개형으로 인체의 뉴런의 작동방식과 유사하여 과거에 activation function으로 많이 사용되었으나 매우 크거나 작은 값에서 gradient vanish 문제가 있고 지수 계산에 연산이 오래걸려 최근에는 잘 쓰이지 않는다.

$$S(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1}.$$

Tanh

$$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Sigmoid 와 유사하나, sigmoid 의 범위가 (0,1)인 반면 tanh 는 (-1,1)이다.

ReLU

f(x)=max(0,x)

최근에 자주 사용되는 activation function 이다. 연산이 간단하여 학습이 빠르고 gradient vanishing 문제를 해결 했다. 함수값이 0 보다 작으면 0 이되어 뉴런이 죽는 현상이 나타나기도 한다.

leaky ReLU

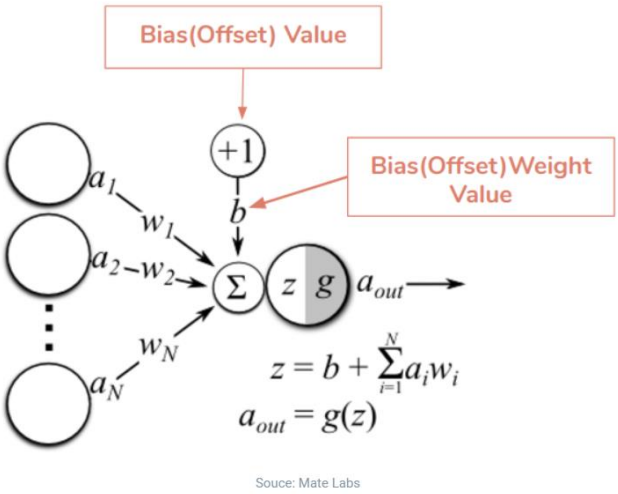
f(x)=max(0.01x, x)

ReLU 에서 0 보다 작은구간에서 0 이 되어 뉴런이 죽는 현상을 해결하기 위해 0 보다 작은 구간에서 작은 기울기 값을 추가하였다.

3. **(Explain MLP)** Describe MLP formula (e.g. 2 layers) with respect to- weight and bias, and explain training process (backpropagation)

$$Y = \sum (weight * input) + bias$$

MLP 에서 다음노드의 값은 이전 노드들의 가중치를 곱한 값과 bias 를 더해서 나타난다.



Souce: Mate Labs

Backpropagation 은 신경망을 학습시킬 때 사용되는 알고리즘인데, 학습진행 시 데이터 input 에 대해 신경망을 통해 output 을 구하고 주어진 정답 label 과 비교하여 cost 를 계산하고 weight, bias 등 모델에 사용되는 parameter 를 업데이트 하는데, 이때 weight 를 업데이트 하는 방법으로 cost 에 대한 parameter 의 변화율의 크기에 비례해 업데이트 하게 되는데 Backpropagation 끝단부터 chain rule 을 이용하여 노드의 변화율을 전달하여 한번의 곱셈만으로 각 노드의 parameter 에 대한 변화율을 계산할 수 있도록 해준다.

$$\frac{\partial E}{\partial w_{ij}^k} = \frac{\partial E}{\partial a_j^k} \frac{\partial a_j^k}{\partial w_{ij}^k},$$

왼쪽 식과 같이 이전 노드까지의 변화율을 안다면 다음 노드에 대한 변화율을 계산하기 용이하다. 이런식으로 gradient 를 역방향으로 전달하여 cost 에 대한 해당 parameter 의 변화율을 쉽게 구하도록 한다.

이렇게 parameter 를 업데이트하는 과정을 신경망을 학습하는 과정이다.

4. (Training and Evaluation)

a. Build a three-layer perceptron. At this time, make the hidden node 1024 dimensions and use ReLU.

b. After the last layer, use softmax and cross-entropy as the loss function.

```
import torch
import torchvision.datasets as dsets
import torchvision.transforms as transforms

device = 'cuda' if torch.cuda.is_available() else 'cpu'
torch.manual_seed(123)
if device == 'cuda':
    torch.cuda.manual_seed_all(123)

learning_rate = 0.1
training_epochs = 30
batch_size = 100

mnist_train = dsets.MNIST(root='./data',
                          train=True,
                          transform=transforms.ToTensor(),
                          download=True)

train_set, val_set = torch.utils.data.random_split(mnist_train, [54000,
 6000])

test_set = dsets.MNIST(root='./data',
                         train=False,
                         transform=transforms.ToTensor(),
                         download=True)


val_dataloader = torch.utils.data.DataLoader(dataset=val_set, batch_siz
e=6000)
test_X= test_set.test_data.view(-1, 28 * 28).float().to(device)
test_Y=test_set.test_labels.to(device)
val_X, val_Y = next(iter(val_dataloader))
val_X=val_X.view(-1, 28 * 28).float().to(device)
val_Y=val_Y.to(device)

data_loader = torch.utils.data.DataLoader(dataset=train_set,
                                          batch_size=batch_size,
                                          shuffle=True,
                                          drop_last=True)

linear1 = torch.nn.Linear(784, 1024, bias=True)
linear2 = torch.nn.Linear(1024, 10, bias=True)
```

```
relu = torch.nn.ReLU()

torch.nn.init.kaiming_uniform_(linear1.weight)
torch.nn.init.kaiming_uniform_(linear2.weight)



model = torch.nn.Sequential(linear1, relu, linear2).to(device)
criterion = torch.nn.CrossEntropyLoss().to(device)    # Softmax is inte
rnally computed.
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
total_batch = len(data_loader)


cost_list=[]
testacc_list=[]
valacc_list=[]

for epoch in range(training_epochs):
    avg_cost = 0

    for X, Y in data_loader:
        X = X.view(-1, 28 * 28).to(device)
        Y = Y.to(device)
        optimizer.zero_grad()
        hypothesis = model(X)
        cost = criterion(hypothesis, Y)
        cost.backward()
        optimizer.step()

        avg_cost += cost / total_batch

    with torch.no_grad():

        prediction = model(val_X)
        correct_prediction = torch.argmax(prediction, 1) == val_Y
        val_accuracy = correct_prediction.float().mean()*100

        prediction = model(test_X)
        correct_prediction = torch.argmax(prediction, 1) == test_Y
        test_accuracy = correct_prediction.float().mean()*100

    cost_list.append(cost)
    testacc_list.append(test_accuracy)
    valacc_list.append(val_accuracy)
    print(f'Epoch:{epoch + 1:3d}, cost = {avg_cost:.9f}, test_acc:{test
_accuracy}, val_acc:{val_accuracy}')
```

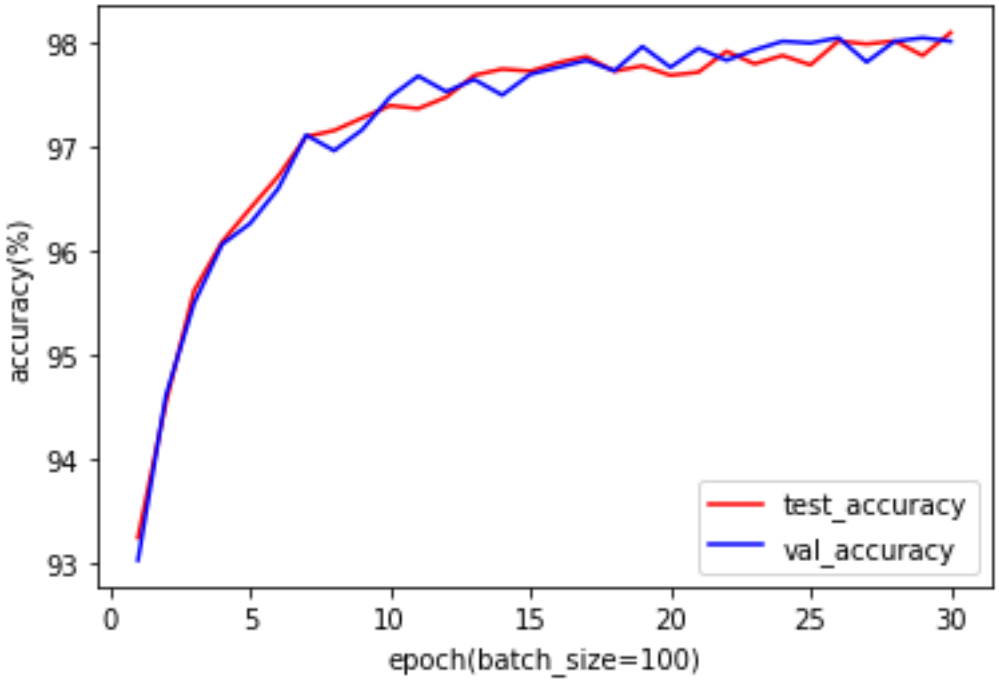
c. Graph validation accuracy and test accuracy for each epoch, and check when convergence occurs.

Batch_size=100, total_epochs=30 으로 진행했다.

Optimizer 로 기본적인 SGD 를 사용했고 learning late=0.1,

ReLu 를 activation function 으로 써서 kaiming_uniform 으로 initialize 했다.

25 epoch 이상에서 정확도가 97.6~98.1 사이에서 진동한다. 이 부근에서 수렴한다고 볼 수 있다.




```
Epoch:  1, cost = 0.390603870, test_acc:93.25,
val_acc:93.03333282470703
Epoch:  2, cost = 0.219631597, test_acc:94.55000305175781,
val_acc:94.61666870117188
Epoch:  3, cost = 0.171263725, test_acc:95.62000274658203, val_acc:95.5
Epoch:  4, cost = 0.140357733, test_acc:96.09000396728516,
val_acc:96.06666564941406
Epoch:  5, cost = 0.118595928, test_acc:96.41000366210938,
val_acc:96.26667022705078
Epoch:  6, cost = 0.102920912, test_acc:96.72000122070312,
val_acc:96.5999984741211
Epoch:  7, cost = 0.090351380, test_acc:97.0999984741211,
val_acc:97.11666870117188
Epoch:  8, cost = 0.080163866, test_acc:97.15999603271484,
val_acc:96.96666717529297
```

```
Epoch:  9, cost = 0.072188608, test_acc:97.27999877929688,
val_acc:97.16667175292969
Epoch: 10, cost = 0.065125756, test_acc:97.39999389648438,
val_acc:97.48332977294922
Epoch: 11, cost = 0.058981914, test_acc:97.3699951171875,
val_acc:97.68333435058594
Epoch: 12, cost = 0.053383891, test_acc:97.47999572753906,
val_acc:97.53333282470703
Epoch: 13, cost = 0.049140517, test_acc:97.68999481201172,
val_acc:97.64999389648438
Epoch: 14, cost = 0.044895712, test_acc:97.75, val_acc:97.5
Epoch: 15, cost = 0.041371234, test_acc:97.72999572753906,
val_acc:97.69999694824219
Epoch: 16, cost = 0.038185138, test_acc:97.80999755859375,
val_acc:97.76667022705078
Epoch: 17, cost = 0.035208419, test_acc:97.8699951171875,
val_acc:97.83333587646484
Epoch: 18, cost = 0.032779373, test_acc:97.72999572753906,
val_acc:97.73332977294922
Epoch: 19, cost = 0.030183259, test_acc:97.77999877929688,
val_acc:97.96666717529297
Epoch: 20, cost = 0.028145077, test_acc:97.68999481201172,
val_acc:97.76667022705078
Epoch: 21, cost = 0.026092904, test_acc:97.7199935913086,
val_acc:97.94999694824219
Epoch: 22, cost = 0.024458682, test_acc:97.91999816894531,
val_acc:97.83333587646484
Epoch: 23, cost = 0.022719920, test_acc:97.79999542236328,
val_acc:97.93333435058594
Epoch: 24, cost = 0.021298053, test_acc:97.87999725341797,
val_acc:98.01667022705078
Epoch: 25, cost = 0.020051252, test_acc:97.79000091552734, val_acc:98.0
Epoch: 26, cost = 0.018833568, test_acc:98.0199966430664,
val_acc:98.04999542236328
Epoch: 27, cost = 0.017673496, test_acc:97.98999786376953,
val_acc:97.81666564941406
Epoch: 28, cost = 0.016663929, test_acc:98.0199966430664,
val_acc:98.01667022705078
Epoch: 29, cost = 0.015698615, test_acc:97.87999725341797,
val_acc:98.04999542236328
Epoch:  30,  cost  =  0.014809718,  test_acc:98.0999984741211,
val_acc:98.01667022705078
```

d. Consider ways to increase the performance of the designed MLP.

데이터를 더 구하여 데이터셋을 늘려본다.
Optimzer 를 SGD 가 아닌 다른 것으로 바꾸어 본다.
Nonlinear 한 성질을 더 추가하기 위해 더 깊게 층을 쌓아 본다.
가중치 제한, Dropout 같은 기법을 이용해 과적합을 막아본다.
CNN 처럼 이미지에 특화된 신경망 구조로 바꾸어 본다.