# KECE470: Pattern Recognition

**School of Electrical Engineering, KOREA UNIVERSITY**

## (Homework #4) Deep Neural Networks

**Report containing the code, results, discussions**

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. **(Build the CNN Model)** Next, you will build 2 kinds of Convolutional Neural Network
   A Network: Conv-Pool-Conv-Pool
   B Network: Conv-Conv-Pool-Conv-Conv-Pool,
   the architectures are as follows and you will take ReLU as the activation function in these two networks,

| A network | | |
|---|---|---|
| Layer | Number of Filters | Padding |
| Input Image | - | - |
| Conv2d (f=3, s=1) | 32 | Valid |
| MaxPool(f=2, s=2) | - | Valid |
| Conv2d (f=3, s=1) | 64 | Valid |
| MaxPool (f=2, s=2) | - | Valid |
| Conv2d (f=3, s=1) | 128 | Valid |
| MaxPool (f=2, s=2) | - | Valid |
| Flatten | - | - |
| Dense | - | - |
| Softmax | - | - |

| B network | | |
|---|---|---|
| Layer | Number of Filters | Padding |
| Input Image | - | - |
| Conv2d (f=3, s=1) | 16 | same |
| Conv2d (f=3, s=1) | 16 | same |
| MaxPool(f=2, s=2) | - | Valid |
| Conv2d (f=3, s=1) | 32 | Valid |
| Conv2d (f=3, s=1) | 32 | Valid |
| MaxPool (f=2, s=2) | - | Valid |
| Flatten | - | - |
| Dense | - | - |
| Softmax | - | - |

```
class A_CNN(torch.nn.Module):
    def __init__(self):
        super(A_CNN, self).__init__()
        self.name='A_Network'
        self.layer1 = torch.nn.Sequential(
            torch.nn.Conv2d(1, 32, kernel_size=3, stride=1, padding='va
lid'),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2))

        self.layer2 = torch.nn.Sequential(
            torch.nn.Conv2d(32, 64, kernel_size=3, stride=1, padding='v
alid'),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2))

        self.layer3=torch.nn.Sequential(
```

```
            torch.nn.Conv2d(64, 128, kernel_size=3, stride=1, padding='
valid'),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2))


        self.fc = torch.nn.Linear(1600, 10, bias=True)
        # torch.nn.init.xavier_uniform_(self.fc.weight)
        # torch.nn.init.normal_(self.fc.weight)

    def forward(self, x):
        out = self.layer1(x)    #cov2d, Relu, maxpool
        out = self.layer2(out) #cov2d, Relu, maxpool
        out = out.view(out.size(0), -1)   #  Flatten
        out = self.fc(out)  #dense
        return out


class B_CNN(torch.nn.Module):
    def __init__(self):
        super(B_CNN, self).__init__()
        self.name='B_Network'
        self.layer1= torch.nn.Sequential(
            torch.nn.Conv2d(1,16,kernel_size=3,stride=1,padding='same')
,
            torch.nn.ReLU(),
            torch.nn.Conv2d(16,16,kernel_size=3,stride=1,padding='same'
),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2,stride=2)
        )

        self.layer2 = torch.nn.Sequential(
            torch.nn.Conv2d(16, 32, kernel_size=3, stride=1, padding='v
alid'),
            torch.nn.ReLU(),
            torch.nn.Conv2d(32, 32, kernel_size=3, stride=1, padding='v
alid'),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2)
        )
        self.fc = torch.nn.Linear(800, 10, bias=True)
        # torch.nn.init.xavier_uniform_(self.fc.weight)
        # torch.nn.init.normal_(self.fc.weight)

    def forward(self, x):
        out = self.layer1(x)  # cov2d, Relu, cov2d, Relu, maxpool
        out = self.layer2(out) # cov2d, Relu, cov2d, Relu, maxpool
```

```
        out = out.view(out.size(0), -1)  # Flatten
        out = self.fc(out) #Dense
        return out
```

softmax layer 는 crossentropy loss 를 계산하는 과정에 포함되어 있다.
Optimizer 는 Adam 을 이용했다.

기타 train 시 사용했던 parameter 는 다음과 같다.

```
learning_rate = 0.001
training_epochs = 15
batch_size = 100
```

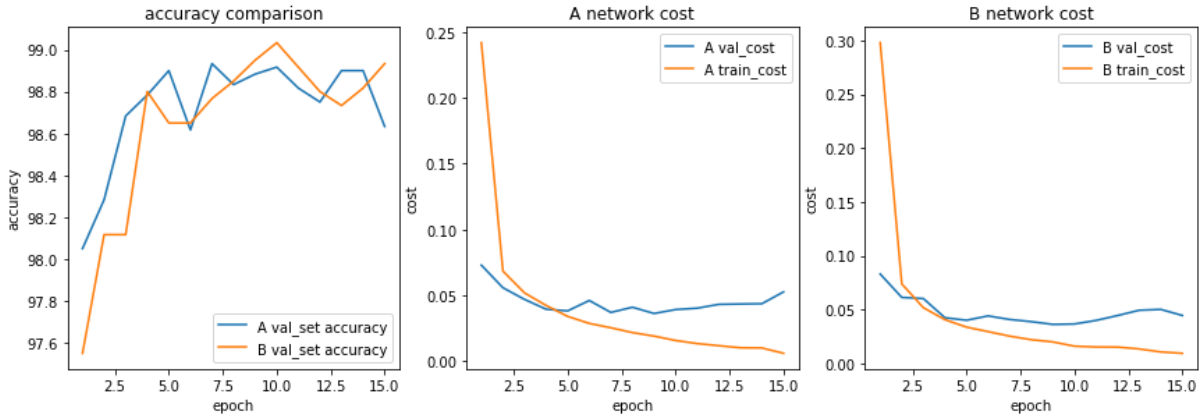
## 3. (Training and Evaluation)

Training 결과

```
A_Network start.
Epoch:   1, train_cost: 0.241849795, val_cost:0.072984867,
val_accuracy:98.04999542236328
Epoch:   2, train_cost: 0.068465360, val_cost:0.055817794,
val_accuracy:98.28333282470703
Epoch:   3, train_cost: 0.051910270, val_cost:0.046995375,
val_accuracy:98.68333435058594
Epoch:   4, train_cost: 0.042391866, val_cost:0.039518513,
val_accuracy:98.78333282470703
Epoch:   5, train_cost: 0.034068305, val_cost:0.038277075,
val_accuracy:98.89999389648438
Epoch:   6, train_cost: 0.028750021, val_cost:0.046153281,
val_accuracy:98.61666870117188
Epoch:   7, train_cost: 0.025445387, val_cost:0.037097197,
val_accuracy:98.93333435058594
Epoch:   8, train_cost: 0.021781098, val_cost:0.041038588,
val_accuracy:98.83333587646484
Epoch:   9, train_cost: 0.019141197, val_cost:0.036287006,
val_accuracy:98.88333129882812
Epoch:  10, train_cost: 0.015790571, val_cost:0.039271526,
val_accuracy:98.91666412353516
Epoch:  11, train_cost: 0.013451847, val_cost:0.040315695,
val_accuracy:98.81666564941406
Epoch:  12, train_cost: 0.011825031, val_cost:0.043228120,
val_accuracy:98.75
Epoch:  13, train_cost: 0.010271395, val_cost:0.043502130,
val_accuracy:98.89999389648438
Epoch:  14, train_cost: 0.010101959, val_cost:0.043717612,
val_accuracy:98.89999389648438
Epoch:  15, train_cost: 0.006074529, val_cost:0.052700948,
val_accuracy:98.63333129882812

B_Network start.
Epoch:   1, train_cost: 0.297926515, val_cost:0.083010308,
val_accuracy:97.54999542236328
Epoch:   2, train_cost: 0.073832929, val_cost:0.061273683,
val_accuracy:98.11666870117188
Epoch:   3, train_cost: 0.051729422, val_cost:0.060200207,
val_accuracy:98.11666870117188
Epoch:   4, train_cost: 0.040568274, val_cost:0.042404115,
val_accuracy:98.79999542236328
Epoch:   5, train_cost: 0.033686411, val_cost:0.040054061,
val_accuracy:98.64999389648438
Epoch:   6, train_cost: 0.029484833, val_cost:0.044097763,
val_accuracy:98.64999389648438
Epoch:   7, train_cost: 0.025299300, val_cost:0.040773939,
val_accuracy:98.76667022705078
Epoch:   8, train_cost: 0.021979678, val_cost:0.038748432,
val_accuracy:98.8499984741211
Epoch:   9, train_cost: 0.019925270, val_cost:0.036213189,
val_accuracy:98.94999694824219
Epoch:  10, train_cost: 0.015946127, val_cost:0.036587216,
val_accuracy:99.03333282470703
```

```
Epoch:  11, train_cost: 0.015256975, val_cost:0.039888274,
val_accuracy:98.91666412353516
Epoch:  12, train_cost: 0.015134245, val_cost:0.044443380,
val_accuracy:98.79999542236328
Epoch:  13, train_cost: 0.013384528, val_cost:0.049358744,
val_accuracy:98.73332977294922
Epoch:  14, train_cost: 0.010607963, val_cost:0.050124820,
val_accuracy:98.81666564941406
Epoch:        15,    train_cost:    0.009362749,    val_cost:0.044513211,
val_accuracy:98.93333435058594
```




a) Describe which network is better? (Analyzing from the accuracy on the validation setset)

위에 그래프를 보면 초반에 A 그래프에 비해 B 그래프가 학습이 느리지만 후반부로 갈수록 B 그래프의 정확성이 평균적으로 높다. 특히 cost 그래프면 epoch 가 10 인부근에서 validation_set 의 코스트가 최소인데 10epoch 부근에서 B 그래프의 정확성이 더 높기에 B network 가 조금 더 낫다.

b) For the network you choose, plot the loss curves on training set and validation set, observe and explain their trend.

B network 를 골랐지만 둘 다 그래프를 그려보았다 두 그래프다 epoch 가 10 인지점에서 최소 validation loss 를 갖는다. Train loss 는 계속해서 감소하는 반면에, validation loss 는 감소하다가 다시 증가하는 U 자형 모습을 보인다.

c) Evaluate the trained network which you choose on the test set, print the accuracy.

```
A test accuracy: 97.419998 , B test accuracy:99.02999
```

B network 를 선택했지만 두 network 에 대해 전부 accuracy 를 측정해 보았다. A network 에 비해 B network 의 정확성이 더 높았고, B network 는 validation set 과 testset 의 정확성 차이가 크지 않은 반면에 A 는 val_set 에 비해 정확성이 더 떨어지는 것을 보아 과적합이 이루어지기 더 쉬운 network 로 보인다.

d) Explain how to prevent overfitting problem when training the deep network.

Validation set 의 코스트가 증가하는 구간에서 학습을 중단한다.

Dropout 같은 기법을 이용해 overfitting 을 방지한다.

Train 에 사용되는 데이터양을 늘린다.

가중치를 regularization 하여 특정 가중치가 과대하게 작용하는 것을 방지한다.