CONTENTS

ELM (Extreme Learning Machine)

4.5 Extreme Learning Machine(ELM)을 이용한 학습 방법

CNN을 사용한 차선 검출은 기존의 복잡한 환경을 학습할 수 있는 방법이긴 했으나 문제점 중 하나인 연산속도 때문에 빠르게 결과를 얻기가 어려웠다. 따라서 이러한 문제를 해결하기 위해서 다르게 접근하고자 하는 방법이 ELM이다.

ELM 이라는 기법은 **Pseudo Inverse** 를 이용하여 구하고자 하는 값의 근사치를 계산하는 방식이다.

우측의 수식을 우리가 CNN 에서 사용하는 “Y = WX” 로 바꿔서 생각을 해보면 이해가 쉽다.

우리는 보통 Gradient decent 를 이용하여 W 값을 구하게 된다.

하지만 Pseudo Inverse 를 이용하여 $W = Y^{+} X$ 로 Matrix 연산을 하게 될 경우 Y(label), X (입력 이미지) 가 정해져 있기 때문에 쉽게 W 의 값을 구할 수 있다.

$$AX=B \quad \text{--- (3)}$$

이 때, m = n이고 A의 역행렬이 존재할 경우 위 선형방정식 (1)을 만족하는 해는 다음과 같이 유일하게 결정된다.

$$X=A^{-1}B \quad \text{--- (4)}$$

그러나, 실제로 부딪히는 대부분의 문제는 m>n인 경우로서 A의 역행렬도 식 (1)을 만족하는 해도 존재하지 않는다. 이 경우에는 다음과 같이 A의 의사역행렬(pseudo inverse)를 이용하여 X를 근사적으로 구한다.

$$X=A^{+}B \quad \text{--- (5)}$$

이 때, A의 pseudo inverse A^{+}는 다음과 같이 계산된다.

$$A^{+}=(A^{T}A)^{-1}A^{T} \quad \text{--- (6)}$$

ELM (Extreme Learning Machine)

$$\beta = H^{+}T \quad (9)$$

따라서 실제 계산 과정은 다음과 같이 진행하면 된다.

(1) 초기에 학습 데이터는 다음과 같이 나온다. 입력 데이터는 $x_1 = [x_{i1}, x_{i2}, \cdots, x_{iN}]$, 타겟 데이터는 $t_1 = [t_{i1}, t_{i2}, \cdots, t_{iN}]$ 활성화 함수(activation function)는 $g(x)$ 데이터 개수는 n이다.

(2) 입력 가중치인 w_i와 b_i는 임의로 값을 설정한다.

(3) 그다음 임의로 준 w_i와 b_i을 가지고 은닉층인 H를 구한다.

(4) 마지막으로 의사역행렬을 사용하여 출력 가중치인 β를 구한다.

따라서 이러한 계산 과정을 거쳐서 가중치를 계산하게 되며, 실제 이러한 방법으로 구했을 시 기존의 역전파 방식보다 훨씬 빠르게 가중치를 계산할 수 있게 된다.

실제 논문에서도 **Pseudo Inverse** 를 이용하여 B(weight)를 구하고 있다.

여기까지는 특별한 것이 없다.

표현하게 된다. 따라서 다음 그림 3과 같이 데이터와 구조가 나타난다. 여기서 w는 실제 입력 가중치로 초기에 임의로 설정하고, β의 경우에는 출력 가중치로 에러가 최소가 될 수 있도록 하기 위해 의사역행렬(pseudo inverse)를 사용하여 최적의 출력 가중치를 구하게 된다. 즉 연산상으로 한 번의 연산으로 출력 가중치를 얻게 되므로 연산량이 매우 적고 빠른 학습에 사용되는 방법이다.

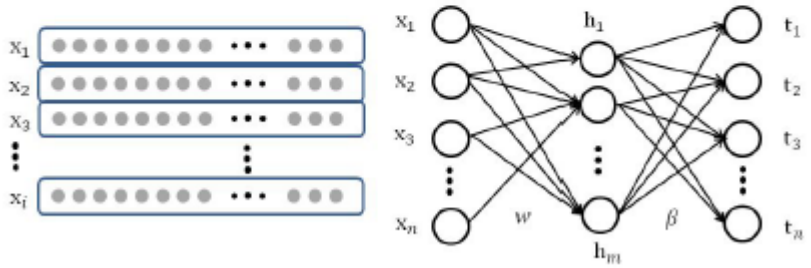


그림 3. 입력, 출력 데이터의 구조 및 ELM의 구조.

ELM layer 에서 pseudo inverse 를 이용해 구할 w 를 포함해 모든 w 값을 랜덤 값으로 사용한다.

심지어 ELCNN (ELM + CNN) 에서도 convolution layer 를 포함해 모든 w 값을 임의로 설정된 값을 사용한다.

ELM (Extreme Learning Machine)

```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5, padding=1)
        self.conv2 = nn.Conv2d(10, 450, kernel_size=4, padding=1)
        self.fc2 = nn.Linear(6*6*450, 10, bias=False)

    def forward(self, x):
        x = self.conv1(x)
        x = F.max_pool2d(x,kernel_size=2)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.max_pool2d(x,kernel_size=2)
        x = F.relu(x)

        x = x.view(-1, self.num_flat_features(x))

        x = self.fc2(x)

        return x
```

```python
def train(self, inputs, targets):
    # print('########### 2) params = ', np.array(self.params).shape)
    # print(self.params[len(self.params) - 1])

    oneHotTarget        = self.oneHotVectorize(targets=targets)
    dimInput            = inputs.size()[1]
    # print('2) dimInput  = ', dimInput)

    xtx= torch.mm(inputs.t(), inputs)   # ATA
    # print(xtx.size(), ' = ', inputs.t().size(), ' X ', inputs.size())

    I = Variable(torch.eye(dimInput))
    # print('I = ', I.size())

    self.M = Variable(torch.inverse(xtx.data + self.C * I.data))    # (ATA)-1
    w = torch.mm(self.M, inputs.t())    # (ATA)-1AT
    # print('1) w = ',  w.size())
    w = torch.mm(w, oneHotTarget)       # A+B = X
    # print('2) w = ', w.size())

    self.w.data = w.t().data
    # print('3) w = ', self.w.size())
```

MNIST 예제를 사용하였고,

위와 같이 간단한 CNN 네트워크 구성 후 Convolution 마지막 계층과 FC2 사이에 ELM 을 추가해 pseudo Inverse 방식으로 W(weight) 값을 구해보면,

약 230 초 만에 Accuracy = 99% 라는 놀라운 결과가 나온다.

```
Start!!
shape of image =  torch.Size([60000, 1, 28, 28])

Test set accuracy: 9874/10000 (99%)

231.26824522018433
```

하지만,
ELCNN 저자는 이것에 너무 집착한 나머지 큰 실수를 한다.

ELCNN

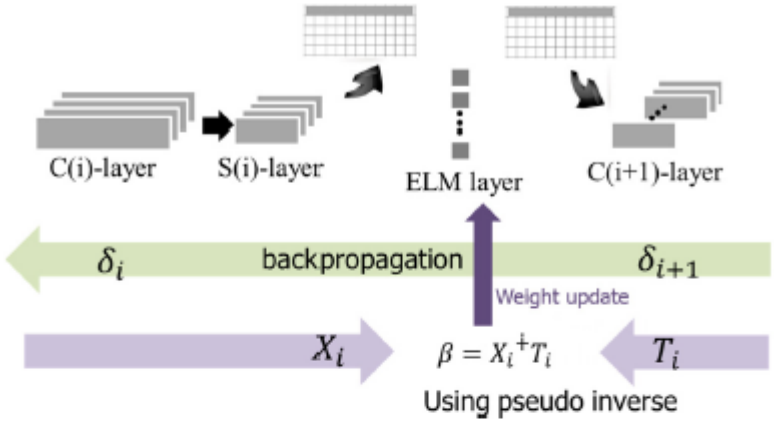


Fig. 6. Structure of the ELM weight updates.

can be further calculated by

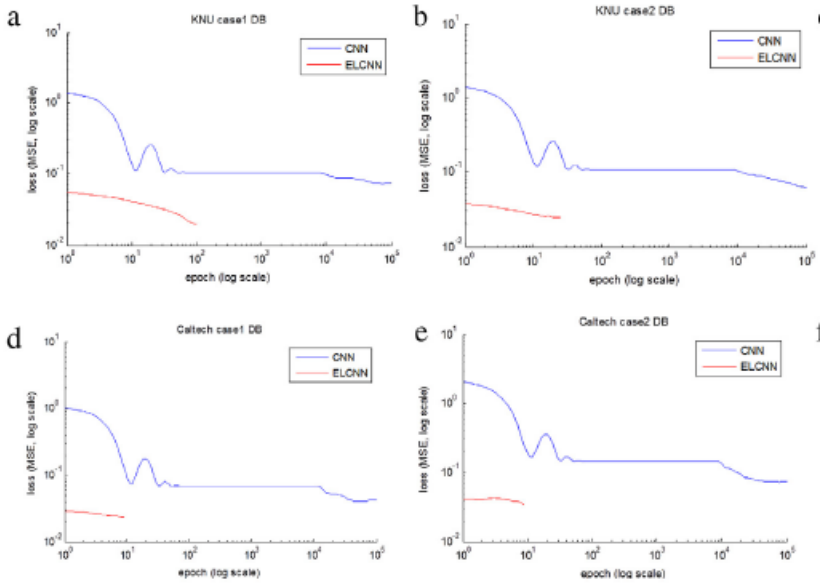
$$\boldsymbol{\beta}_i = \boldsymbol{K}_i^{\dagger} \boldsymbol{T}_i. \tag{16}$$

Thus, we can summarize the entire ELCNN process, which is illustrated in Fig. 6:

1. Compute forward pass.
2. Compute error of convolutional layers using backpropagation.
3. Calculate target of ELM1 with respect to the backpropagated error.
4. Calculate ELM1 weights to minimize the forward pass error.
5. Calculate target of ELM2 with respect to the backpropagated error
6. Calculate ELM2 weights to minimize the forward pass error.
7. Repeat 2–6.
8. Calculate ELM3.
9. Repeat 1–8 if the error is smaller than a given threshold.

Backpropagation 의 값을 이용한다고 되어 있고,

그 결과를 ELM 에 활용한다고 되어 있다.

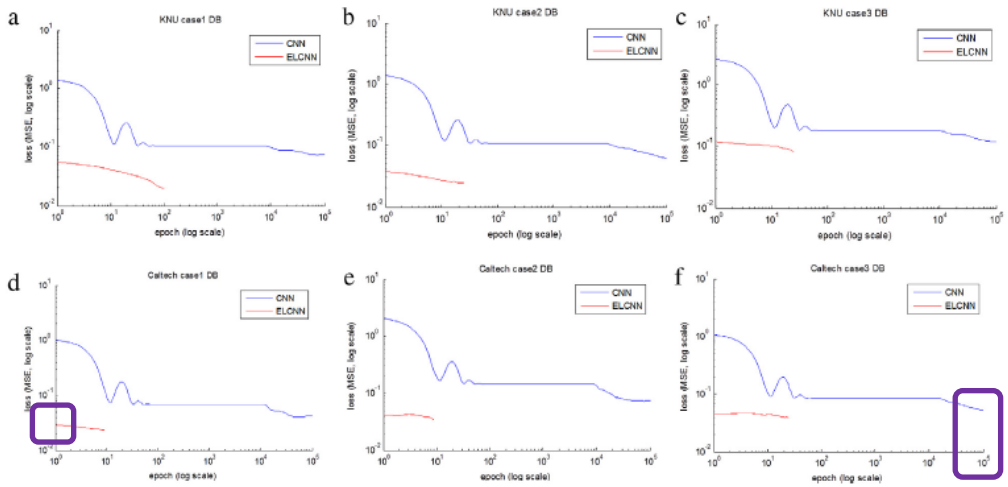


결과도 속도 측면에서 CNN에 비해 ELCNN 이 월등한 것과 같이 나온다.

하지만 여기는 치명적인 함정이 있다.

앞서 이야기 했듯이 ELM 을 활용하여 W 를 근사하기 위해서는 모든 데이터를 한번에 Training 에 활용해야 한다는 것이다.

ELCNN



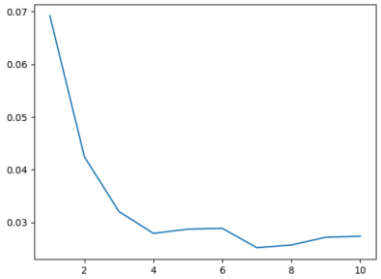
MNIST 트레이닝 데이터 60,000 개를 모두 사용하여,
- CNN 과 ELCNN 돌린 결과가 위와 같다.

일단 붉은색 그래프를 보자.
- 전체 데이터에 대한 근사치를 나타내기 때문에 첫 번째 수행 당시에 벌써 최종 단계에 근접한 결과를 보이며, 매 에폭마다 "back-propagation" 에 의해 convolution layer 의 w 값들이 아주 조금씩 개선되는 모습이다.

파란색 그래프를 마저 보자.
- Gradient decent 를 이용하여 w 을 구하기 때문에 초반에 약간 빠르게 수렴하다가 뒤로 갈 수록 느리게 수렴하는 모습이다.
- 그리고, Batch size = 60,000 이기 때문에 전체적으로 아주 느리게 수렴한다.

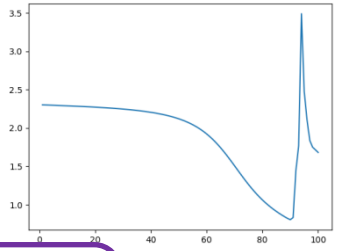
ELCNN 은 ELM 특성 상 초기에 거의 목적지에 와있는 상태이다. 따라서,둘 간의 목적지에 도달하는데 걸리는 시간을 비교한다는 것 자체가 잘못되어 있다.

이제 치명적인 결함이 보이나요?
아래와 같이 CNN (Batch size = 32) 결과를 포함해서 보면 명확한 문제점이 보입니다.

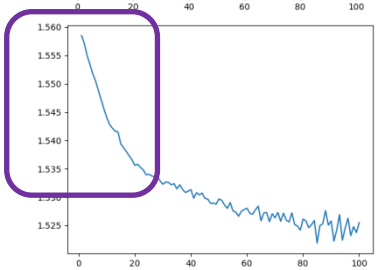


CNN
BatchSize: 32
걸린시간: 3,916초
Loss: 0.0273
ACC: 99.03%

CNN 의 경우 SGD 방식으로 돌리면 10 에폭 만에 원하는 결과에 도달이 가능하다.



CNN
BatchSize: 60,000
걸린시간: 7,824초
Loss: 1.6816
ACC: 67.21%
논문에서와 같이 엄청 오래 돌리면 99% 이상에 근접할 것임



ELCNN
BatchSize: 60,000
걸린시간: 10,143초
Loss: 1.5255
ACC: 98.98%
초반부터 결과가 매우 좋다.

또 하나의 함정은 굳이 ELM의 장점을 부각하고자 CNN 을 batch size =60,000으로 돌릴 필요가 없다는 것이다.

ELCNN

앞선 결과는 글을 읽는 사람으로 하여금 오류에 빠지기 쉽게 결과를 포장해 놓았다.
그리고, 앞선 문제점이 노출되는 것을 꺼려서 이 논문 어디에도 Batch size 에 대한 언급하고 있지 않는 듯 하다.

결론은,
ELCNN 은 이 논문에서 이야기 하고 있는 이점들은 편향된 테스트의 결과이다. 이게 정말 좋았다면... 다들 이걸 쓰고 있을 것이다.

다만,
우리는 또 다른 실험 결과에 주목할 필요가 있다. 아래의 결과와 같이 **CNN 이 차선을 찾기 위한 전처리 과정에서 좋은 결과**를 보인다는 것이다.

Table 2
Performance evaluation in three conditions in two datasets comparing an ELM, a CNN with an ELM classifier, a conventional CNN, and the ELCNN.

Dataset	Case	Method (%)			
		ELM	CNN	CNN with ELM classifier	ELCNN
KNU database	Case 1	59.6	94.8 (±0.39)	96.1 (±0.30)	96.2 (±0.44)
	Case 2	85.4	93.3 (±0.63)	95.0 (±1.20)	95.9 (±1.44)
	Case 3	52.9	92.3 (±1.16)	94.8 (±1.40)	93.0 (±2.78)
Caltech dataset	Case 1	93.8	97.5 (±2.12)	97.5 (±2.32)	98.1 (±1.53)
	Case 2	70.0	95.0 (±0.00)	100.0 (±0.00)	100.0 (±0.00)
	Case 3	72.5	90.0 (±1.44)	95.5 (±2.45)	96.5 (±1.22)

a

b

c

d

ELCNN

이 논문 상에는 자신들이 ELM 을 적용하기 전 BASE 가 되는 네트워크를 설명하고 있다.

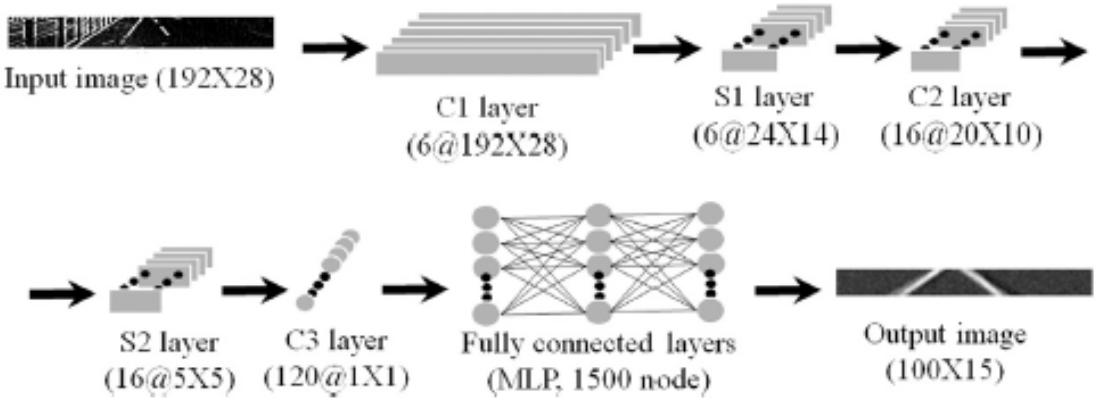


Fig. 1. CNN structure.

우리는 위의 네트워크를 기반으로 하여 데이터를 만들어 테스트를 진행해보면서 더 나은 결과를 위해 batch normalization 등의 추가 기법을
도입해보는 것도 좋은 방안일 듯 하다.

대략적인
네트워크 메모리 및 Weight를 계산해보면 우측과 같이 작은 크기의 네트워크가 나오므로 결과만 좋다면 우리가 사용하는 ZCU102 보드에도 충분히 올려볼 수 있을 것으로 보인다.

Input
MEM: 192 X 28 X 1 = 9,216 / W: 0

C1 layer
MEM: 192 X 28 X 6 = 55,296 / W : (3 X 3 X 1) X 6 = 54

S1 layer
MEM: 24 X 14 X 16(concat) X 6 = 32,256 / W : 0

C2 layer
MEM: 20 X 10 X 16 = 3,200
W : 4 X 4 X 16(concat) X 16 = 4,096

S2 layer
MEM: 5 X 5 X 8(concat) X 16 = 3,200 / W : 0

C3 layer
MEM: 1 X 1 X 120 = 120
W : 5 X 5 X 8(concat) X 120 = 24,000

FC layer
MEM: 1 X 1 X 1,500 = 1,500
W : 1 X 1 X 120 X 1,500 = 180,000

Total
MEM: 104,788
W: 208,150

SUM = 312,938 (313K)