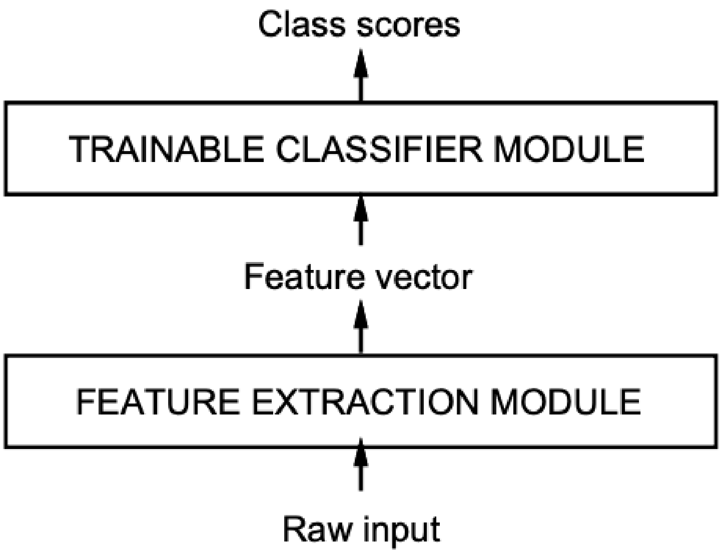
📖

[논문 리뷰] LeNet-5 (1998)

1. Introduction

본 논문의 주된 내용은, hand-designed hueristics에 의존하기보다는, automatic Learning을 사용하는 것이 pattern recognition에 효과적이라는 것이다.

▼ PREVIOUS PROBLEM



1. Hand-designed feature extractor

hand-designed feature extractor는 입력으로부터 관련있는 정보만 수집하고, 무관한 정보를 제거합니다. 이는 사람이 설계한 feature extractor로 추출한 정보만으로 분류기로 전달되므로, limited learning만이 될 수 밖에서 없습니다. 그러므로, feature extractor된 그 자체로 학습해야 합니다.

2. Using Many Parameters

하나의 이미지는 몇 백개의 변수(pixel)을 포함하고 있습니다. 또한 fully connected multi layer의 첫 번째 layer에 이미 몇 만개의 가중치를 가지고 있습니다. 이러한 parameters는 Memory의 저장공간을 많이 필요하게 합니다.

3. Ignore Input topology

이미지는 2D 구조(if not using channel)를 가지고 있으므로, 인접한 변수(pixel)들은 공간적으로 매우 큰 상관관계를 가지고 있습니다. 하지만, fully-connected-multi-layer는 인접한 변수들에 대하여 공간적인 정보를 얻지 못하는 문제가 있습니다.

▼ 2. Learning From Data

Automatic ML에는 여러가지 approaches가 존재하지만, most succesful는 gradient-based-learning이다.

$Y^p = F(Z^p, W)$

- Z^p 는 p번째 input, W는 parameter이다.
- Y^p 는 Z^p의 class label or related about class propability

E^p = $D(D^p, F(W, Z^p))$는 D^p와 Z^p로 계산한 결과의 일치하지 않는 정도를 계산

- Mean Squared Error(MSE) : $E_{train}(W)$는 E^p의 평균이다.
- The learning problems consists in finding minimizes of W $E_{train}(W)$

$E_{test} - E_{train} = k(h/P)^{\alpha}$

- P는 training sample이고, h는 time complex이다.
- h의 최적의 값은 E_{test}의 일반화 오류(error)를 최소화하는 것이다.

⇒ Input과 parameter를 통하여, class별 점수를 산출하고, test loss와 train loss의 차이를 감소시켜야함

▼ 3. Gradient-Based Learning

parameter의 관점에서 식을 최소화하는 것이 CS에서 근본적으로 발생하는 문제이다.

Gradient-Based Learning은 일반적으로, 미분값을 최소화하는 것을 더욱 쉽게합니다.

A popular minimization procedure is the stochastic gradient algorithm이다. 이것은 noisy parameter vector을 사용함으로서, 업데이트 됩니다.

$$W_k = W_{k-1} - \epsilon \frac{\partial E^{p_k}(W)}{\partial W}$$

확률적 경사 하강법(Stochastic Gradient Descent)는 일반적인(Gradient Descent)보다

더욱 학습속도가 빠르다.

4. LeNet-5 Architecture

- 저자 표기법
  - Cx : Convolution layer
  - Sx : Subsampling (pooling) layer
  - Fx : fully-connected layer
  - x : index pf the layer

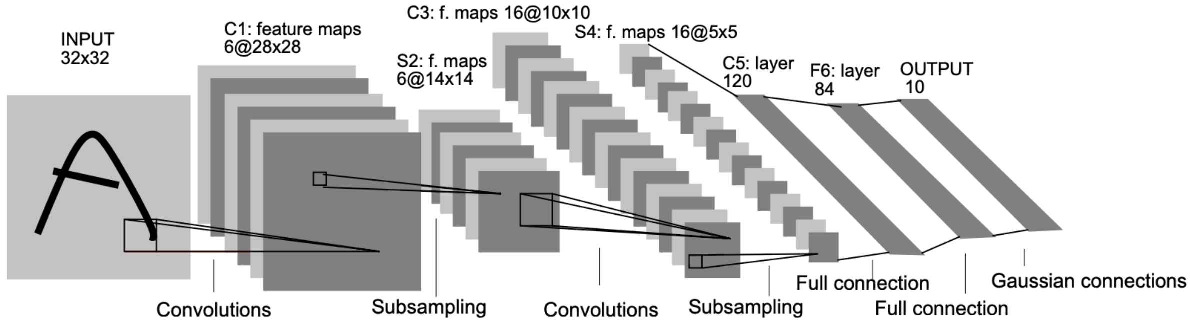


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

LeNet-5는 32x32 크기의 흑백 이미지에서 학습된 7 layer Convolutional Neural Network이다.

[Conv(C1) → SubSampling(S2) → Conv(C3) → SubSampling(S4) → Conv(C5) → FC → FC]

▼ **Layer Structure**

Layer C1

5x5 크기의 kernel 6개와 stride=1을 지닌 convolutional layer이다.

Layer S2

2x2 크기의 kernel 6개와 stride = 2를 지닌 subsampling layer이다.

Layer C3

5x5 크기의 kernel 16개와 stride = 1을 지닌 Convolution layer이다.

C3의 feature map과 연결된 S2의 feature map을 보여줍니다.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

TABLE I

EACH COLUMN INDICATES WHICH FEATURE MAP IN S2 ARE COMBINED BY THE UNITS IN A PARTICULAR FEATURE MAP OF C3.

Layer S4

2x2 크기의 kernel 16개와 stride = 2를 지닌 subsampling layer이다.

Layer C5

5x5 크기의 kernel 120개와 stride = 1을 지닌 convolutional layer이다.

Layer F6

입력 120개에 대한, 출력 84개를 합니다.

Layer F7

입력 84개에 대한 출력, 즉 Output Layer 10개를 진행합니다.

(이때, MSE[Mean Squared Error]) 사용

▼ ONLY MODEL CODE [Pytorch]

```
class LeNet5(nn.Module):
    def __init__(self, n_classes):
        super(LeNet5, self).__init__()

        self.feature_extractor = nn.Sequential(
            nn.Conv2d(in_channels = 1, out_channels = 6, kernel_size = 5, stride = 1),
            nn.Tanh(),
            nn.AvgPool2d(kernel_size = 2),
            nn.Conv2d(in_channels = 6, out_channels = 16, kernel_size = 5, stride = 1),
            nn.Tanh(),
            nn.AvgPool2d(kernel_size = 2),
            nn.Conv2d(in_channels = 16, out_channels = 120, kernel_size = 5, stride = 1),
            nn.Tanh()
        )

        self.classifier = nn.Sequential(
            nn.Linear(in_features = 120, out_features = 84),
            nn.Tanh(),
            nn.Linear(in_features = 84, out_features = n_classes),
        )

    def forward(self, x):
        x = self.feature_extractor(x)
        x = torch.flatten(x , 1)
        logits = self.classifier(x)
        probs = F.softmax(logits, dim = 1)
        return logits, probs
```

```python
#!pip install torchsummary
from torchsummary import summary

summary(model, (1, 32, 32)) # Model, Input_Size
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1            [-1, 6, 28, 28]             156
              Tanh-2            [-1, 6, 28, 28]               0
         AvgPool2d-3            [-1, 6, 14, 14]               0
            Conv2d-4           [-1, 16, 10, 10]           2,416
              Tanh-5           [-1, 16, 10, 10]               0
         AvgPool2d-6             [-1, 16, 5, 5]               0
            Conv2d-7            [-1, 120, 1, 1]          48,120
              Tanh-8            [-1, 120, 1, 1]               0
            Linear-9                   [-1, 84]          10,164
             Tanh-10                   [-1, 84]               0
           Linear-11                   [-1, 10]             850
================================================================
Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.11
Params size (MB): 0.24
Estimated Total Size (MB): 0.35
----------------------------------------------------------------
```

▼ Train & Validation MODEL CODE [Pytorch]

```python
from datetime import datetime

import numpy as np

import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader

from torchvision import datasets, transforms
import matplotlib.pyplot as plt

device = "cuda:0" if torch.cuda.is_avilable() "cpu"
```

```python
# Config Setting
random_seed = 42
learning_rate = 1e-3
batch_size = 32
n_epochs = 15

img_size = 32
n_classes = 10
```

```
def get_accuracy(model, data_loader, device):
  correct_pred = 0
  n = 0

  with torch.no_grad():
    model.eval()
    for X, y_true in data_loader:
      X, y_true = X.to(device), y_true.to(device)

      _, y_pred = model(X)
      _, predicted_labels = torch.max(y_pred, 1)

      n += y_true.size(0)
      correct_pred += (predicted_labels == y_true).sum()

   return correct_pred.float() / n
```

```
def plot_losses(train_losses, valid_losses):
  plt.style.use("seaborn")

  train_losses = np.array(train_losses)
  valid_losses = np.array(valid_losses)

  fig, ax = plt.subplots(figsize =(8, 4.5))
  ax.plot(train_losses, color = 'blue', label = 'train')
  ax.plot(valid_losses, color = 'red', label = 'valid')
  ax.set(title = 'Loss over epochs',
         xlabel = 'Epochs',
         ylabel = 'Loss')
  ax.legend()
  fig.show()
  plt.style.use("default")
```

```
def train(train_loader, model, criterion, optimizer, device):
  model.train()
  running_loss = 0
  for X, y_true in train_loader:
    optimizer.zero_grad()
    X, y_true = X.to(device), y_true.to(device)
    y_hat, _ = model(X)
    loss = criterion(y_hat, y_true)
    running_loss += loss.item() * X.size(0)

    loss.backward()
    optimizer.step()

  epoch_loss = running_loss / len(train_loader.dataset)
  return model, optimizer, epoch_loss
```

```
def validation(valid_loader, model, criterion, device):
  model.eval()
  running_loss = 0
  for X, y_true in valid_loader:
    X, y_true = X.to(device), y_true.to(device)

    y_hat, _ = model(X)
    loss = criterion(y_hat, y_true)
    running_loss += loss.item() * X.size(0)
  epoch_loss = running_loss / len(valid_loader.dataset)

  return model, epoch_loss
```

```
def training_loop(model, criterion, optimizer, train_loader, valid_loader, epochs, device, print_every = 1):
  best_loss = 1e10
  train_losses = []
  valid_losses = []

  for epoch in range(epochs):
    model, optimizer, train_loss = train(train_loader, model, criterion, optimizer, device)
    train_losses.append(train_loss)

    with torch.no_grad():
      model, valid_loss = validation(valid_loader, model, criterion, device)
      valid_losses.append(valid_loss)

    if epoch % print_every == (print_every - 1):
        train_acc = get_accuracy(model, train_loader, deivce)
        valid_acc = get_accuracy(model, valid_loader, device)

    print(f'{datetime.now().time().replace(microsecond=0)} --- '
          f'Epoch: {epoch}\t'
          f'Train loss: {train_loss:.4f}\t'
          f'Valid loss: {valid_loss:.4f}\t'
          f'Train accuracy: {100 * train_acc:.2f}\t'
          f'Valid accuracy: {100 * valid_acc:.2f}')

  plot_losses(train_losses, valid_losses)
  return model, optimizer, (train_losses, valid_losses)
```

```
transform = transforms.Compose([transforms.Resize((32, 32)),
                              transforms.ToTensor()])

train_dataset = datasets.MNIST(root = 'mnist_data',
                              train = True,
                              transform= transform,
                              download = True)

valid_dataset = datasets.MNIST(root = 'mnist_data',
                              train = False,
                              transform = transform,
                              download = True)

train_loader = DataLoader(dataset = train_dataset,
                         batch_size = batch_size,
                         shuffle = True)

valid_loader = DataLoader(dataset = valid_dataset,
                         batch_size = batch_size,
                         shuffle = False)
```

```
class LeNet5(nn.Module):
    def __init__(self, n_classes):
        super(LeNet5, self).__init__()

        self.feature_extractor = nn.Sequential(
            nn.Conv2d(in_channels = 1, out_channels = 6, kernel_size = 5, stride = 1),
            nn.Tanh(),
            nn.AvgPool2d(kernel_size = 2),
            nn.Conv2d(in_channels = 6, out_channels = 16, kernel_size = 5, stride = 1),
            nn.Tanh(),
            nn.AvgPool2d(kernel_size = 2),
            nn.Conv2d(in_channels = 16, out_channels = 120, kernel_size = 5, stride = 1),
            nn.Tanh()
        )

        self.classifier = nn.Sequential(
            nn.Linear(in_features = 120, out_features = 84),
            nn.Tanh(),
            nn.Linear(in_features = 84, out_features= n_classes),
        )
```

```
    def forward(self, x):
        x = self.feature_extractor(x)
        x = torch.flatten(x, 1)
        logits = self.classifier(x)
        probs = F.softmax(logits, dim = 1)
        return logits, probs
```

```
model = LeNet5(n_classes).to(device)
optimizer = torch.optim.Adam(model.parameters(), lr = 1e-3)
criterion = nn.CrossEntropyLoss()

model, optimizer, _ = training_loop(model, criterion, optimizer, train_loader, valid_loader, n_epochs, device)
```

[Reference]

Gradient-based learning applied to document recognition

Multilayer neural networks trained with the back-propagation algorithm constitute the best example of a successful gradient based learning technique. Given an appropriate network architecture, gradient-based learning algorithms can be used to synthesize a

https://ieeexplore.ieee.org/document/726791

http://vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/fb05b096-7f02-45e6-9844-0877a9a6290b/lenet5-baseline.ipynb