DEEP LEARNING
WITH PYTORCH

SOLUTIONS FOR MODELING

SERVICE WITH DL

◆ WEB 기반 서비스 연동

❖ HTTP.Server

- Python에서 프로그램 테스트용으로 제공하는 웹 서버 모듈
- 서버 구동 명령어 : **python –m http.server 8080**

SERVICE WITH DL

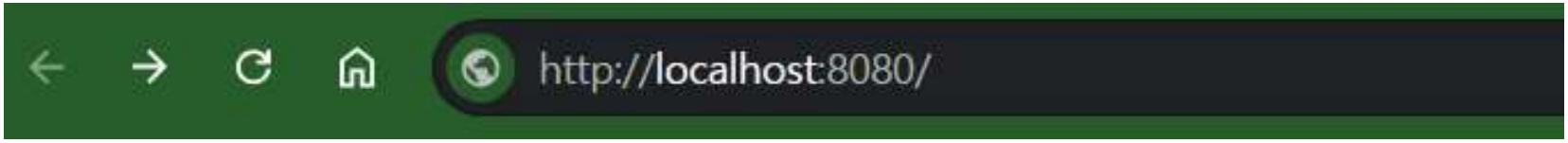
◆ WEB 기반 서비스 연동

❖ HTTP.Server 서버 구동

- 명령어 입력

```
(TORCH_38) C:\Users\anece\TORCH_DL\D0919>python -m http.server 8080
Serving HTTP on :: port 8080 (http://[::]:8080/) ...
```

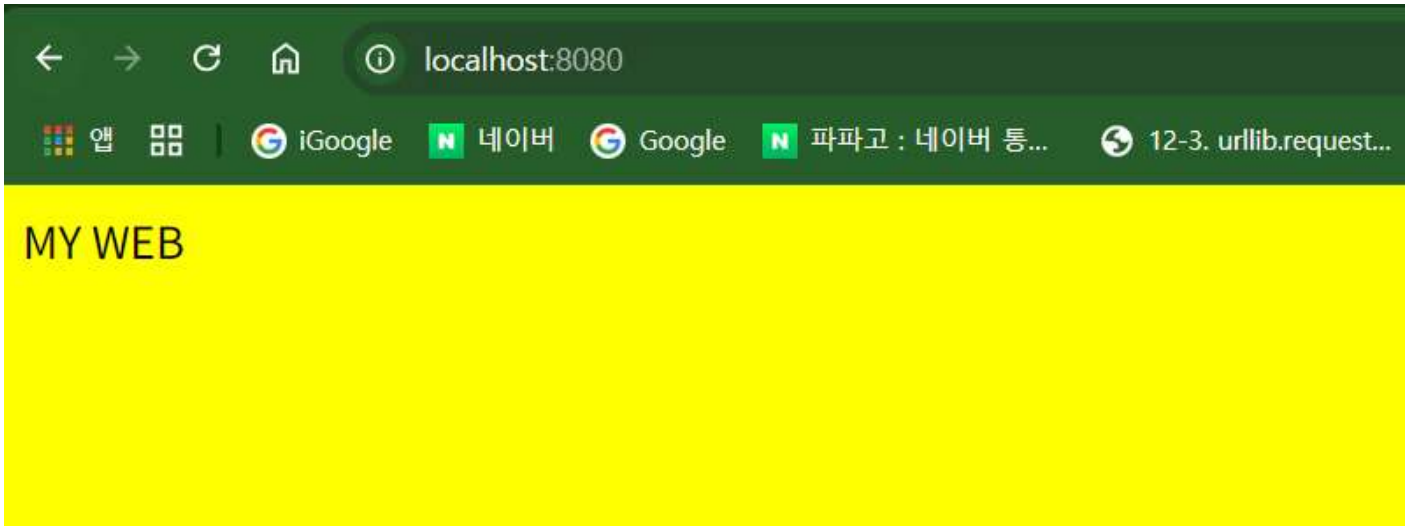
- 브라우저



SERVICE WITH DL

◆ WEB 기반 서비스 연동

❖ HTTP.Server 서버 구동



SERVICE WITH DL

◆ WEB 기반 서비스 연동

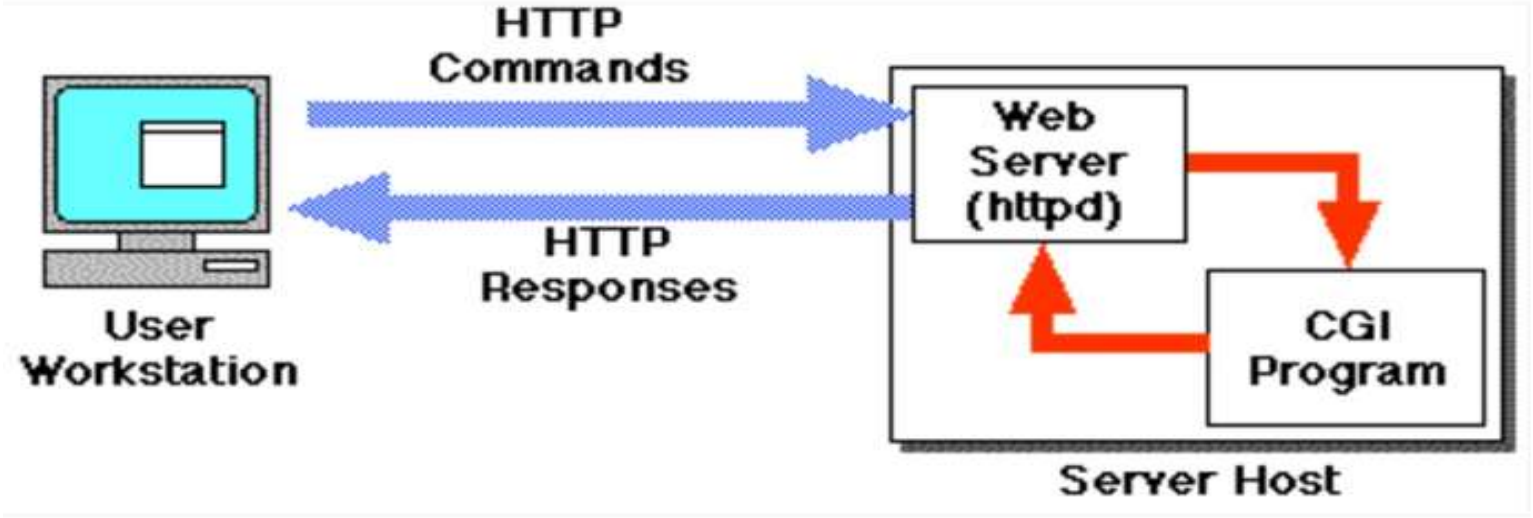
❖ CGI(Command Gateway Interface)

- 동적 웹페이지 요청을 CGI 프로그램으로 전달
- CGI는 적절한 프로그램으로 전달 후 결과 생성
- 생성된 결과를 WEB Server로 전달

SERVICE WITH DL

◆ WEB 기반 서비스 연동

❖ CGI(Command Gateway Interface)

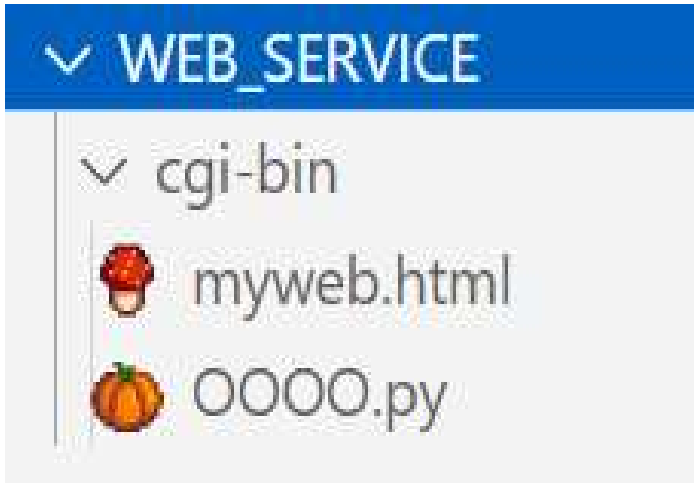


SERVICE WITH DL

◆ WEB 기반 서비스 연동

❖ CGI(Command Gateway Interface)

- cgi-bin 폴더
  - html파일 존재
  - OOO.py 파일 존재 ← 웹 연동 처리
  - 모델 파일 존재



SERVICE WITH DL

◆ WEB 기반 서비스 연동

❖ CGI(Command Gateway Interface)

- 서버 구동 명령어 : **python –m http.server --cgi 8080**

- WEB 연결 : **http://localhost:8080/cgi-bin/OOO.py**

SOLUTIONS FOR MODELING

SOLUTIONS FOR MODELING

◆ DNN MODEL => Layer + AF

LAYER --- 입력층 Full Connected Layer 즉, Linear

LAYER --- 은닉층 Full Connected Layer 즉, Linear

LAYER --- 은닉층 Full Connected Layer 즉, Linear

LAYER --- 은닉층 Full Connected Layer 즉, Linear

LAYER --- 출력층 Full Connected Layer 즉, Linear

최소 2개 이상

SOLUTIONS FOR MODELING

◆ OVERFITTING

▪ 해결방법

- Train Dataset 늘리기 ➔ 쉽지 않음 , 비용, 인력
- Feature의 개수 줄이기 ➔ 저차원
- Regularization(규제) term 추가
- Dropout Layer 추가

SOLUTIONS FOR MODELING

◆ OVERFITTING

▪ 해결방법 - **nn.Dropout Layer**

- 동작 : **일부 노드/퍼셉트론 무작위 비활성화**
- 효과 : **Overfitting 방지**

  매번 다른 형태 노드 학습 진행 >>> **성능 향상** : **앙상블 효과**

SOLUTIONS FOR MODELING

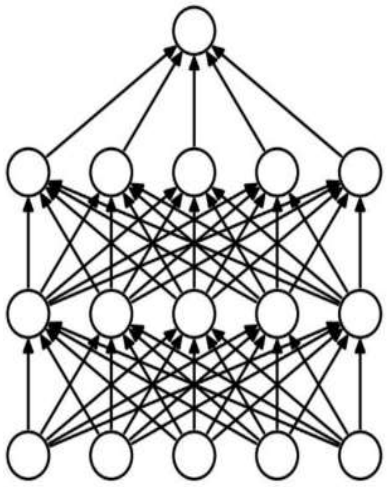
◆ OVERFITTING

▪ 해결방법 - **nn.Dropout Layer**

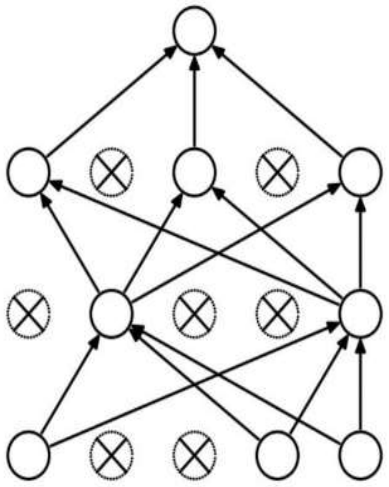
[주의]

- Train Mode : 무작위 선별적 노드
  dropout = True

- Evalu Mode : 모든 노드 사용
  dropout = False



Standard Neural Net



After applying dropout.

SOLUTIONS FOR MODELING

◆ OVERFITTING

▪ 해결방법 - **nn.Dropout Layer**

CLASS torch.nn.Dropout(p=0.5, inplace=False)

- 훈련 중 확률 사용하여 입력 텐서의 일부 요소를 무작위로 0으로 설정
- 연산 진행 되지 않음
- 매개변수
  - p : Layer 내 노드/퍼셉트론이 0이 될 확률 [기본값: 0.5]

SOLUTIONS FOR MODELING

◆ OVERFITTING

▪ 해결방법 - **nn.Dropout Layer**

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(28 * 28, 512)
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, 10)
        self.dropout_prob = 0.5
```

SOLUTIONS FOR MODELING

◆ OVERFITTING

- 해결방법 - **nn.Dropout Layer**

```
def forward(self, x):
        x = x.view(-1, 28 * 28)
        x = F.relu( self.fc1(x) )
        x = F.dropout(x, training = self.training, p = self.dropout_prob)
        x = F.relu( self.fc2(x) )
        x = F.dropout(x, training = self.training, p = self.dropout_prob)
        return self.fc3(x)
```

SOLUTIONS FOR MODELING

◆ OVERFITTING

▪ 해결방법 - **nn.Dropout Layer**

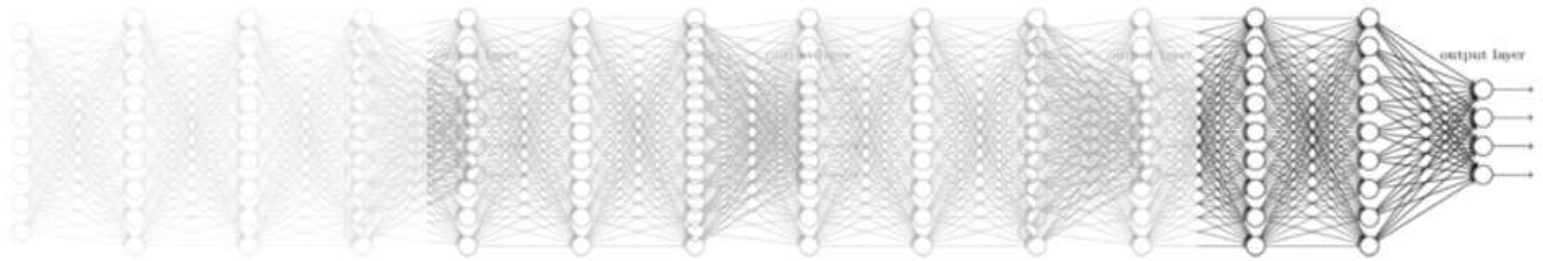
```
class DropoutModel(nn.Module):
    def __init__(self):
        super( ).__init__()
        self.layer1 = nn.Linear(784, 1200)
        self.dropout1 = nn.Dropout(0.5)
        self.layer2 = nn.Linear(1200, 1200)
        self.dropout2 = nn.Dropout(0.5)
        self.layer3 = nn.Linear(1200, 10)
```

```
def forward(self, x):
    x = F.relu(self.layer1(x))
    x = self.dropout1(x)
    x = F.relu(self.layer2(x))
    x = self.dropout2(x)
    return self.layer3(x)
```

SOLUTIONS FOR MODELING

◆ Gradient Vanishing & Exploding

- **역전파 과정에서 출력층에서 멀어질수록 Gradient 값이 매우 작아지는 현상**
- 입력층 가까운 층들에서 가중치 업데이트 제대로 되지 않아 **최적 모델 찾을 수 없음**



SOLUTIONS FOR MODELING

◆ Gradient Vanishing & Exploding

▪ 해결방법

- 활성화 함수 ➔ 입력층/은닉층 ReLu, 출력층 Sigmoid/Softmax
- Normalization ➔ 배치 정규화(batch normalization)
- Weight initialization ➔ 세이비어(Xavie) /글로럿(Glorot) , 헤(He)
- Gradient Clipping ➔ 기울기 폭주 막기 위해 임계값 넘지 않도록 값 자르기

SOLUTIONS FOR MODELING

◆ Gradient Vanishing & Exploding

▪ 해결방법 - **배치 정규화(batch normalization)**

- nn.Linear 모듈 이용하여 입력층/ 출력층 구성
- nn.BatchNorm1d 모듈을 이용하여 은닉층 출력에 배치 정규화 적용
- 은닉층의 출력에 배치 정규화를 적용하여 학습이 더 잘 되도록 구성

SOLUTIONS FOR MODELING

◆ Gradient Vanishing & Exploding

▪ 해결방법 - **배치 정규화(batch normalization)**

torch.nn.BatchNorm1d/2d(num_features, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

- num_features : 입력 데이터의 채널 수 지정. 반드시 지정
- eps : 분모에 더해지는 작은 값, 0으로 나누는 것 방지 위한 인자 [기본값 1e-05]
- momentum : 이전 배치의 평균과 분산값 얼마나 반영할지를 지정 [기본값 0.1].
- affine : 정규화된 값을 확대 및 이동시킬지 여부 지정 인자 [기본값 True]
- track_running_stats : 배치 정규화 효과 추적할지 여부 지정 인자 [기본값 True]

SOLUTIONS FOR MODELING

◆ Gradient Vanishing & Exploding

▪ 해결방법 - **배치 정규화(batch normalization)**

```
class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.fc1 = nn.Linear(100, 50)
        self.bn = nn.BatchNorm1d(num_features=50)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(50, 10)
```

```
def forward(self, x):
        x = self.fc1(x)
        x = self.bn(x)
        x = self.relu(x)
        x = self.fc2(x)
        return x
```

SOLUTIONS FOR MODELING

◆ Gradient Vanishing & Exploding

▪ 해결방법 - **Weight initialization : 세이비어(Xavie) /글로럿(Glorot)**

- uniform, normal 분포로 초기화

Xavier Uniform Initialization

$$W \sim U(-\sqrt{\frac{6}{n_{in}+n_{out}}}, \sqrt{\frac{6}{n_{in}+n_{out}}})$$

Xavier Normal Initialization

$$W \sim N(0, Var),\ Var = \sqrt{\frac{2}{n_{in}+n_{out}}}$$

SOLUTIONS FOR MODELING

◆ Gradient Vanishing & Exploding

- 해결방법 - **Weight initialization : 세이비어(Xavie) /글로럿(Glorot)**

```
import torch
import torch.nn as nn

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()

        self.fc1 = nn.Linear(8, 4)
        self.fc2 = nn.Linear(4, 2)
        self.fc3 = nn.Linear(2, 1)

    def forward(self, x):
        return self.fc3(self.fc2(self.fc1(x)))
```

SOLUTIONS FOR MODELING

◆ Gradient Vanishing & Exploding

▪ 해결방법 - **Weight initialization : 세이비어(Xavie) /글로럿(Glorot)**

```
# fc1 layer에 대한 xavier uniform initialization
nn.init.xavier_uniform_(model.fc1.weight)

# fc1 layer에 대한 xavier normal initialization
nn.init.xavier_normal_(model.fc1.weight)
```

SOLUTIONS FOR MODELING

◆ Gradient Vanishing & Exploding

- 해결방법 - **Weight initialization : 세이비어(Xavie) /글로럿(Glorot)**

```
# 전체 layer에 대하여 xavier uniform initialization 설정
for name, child in model.named_children():
    nn.init.xavier_uniform_(child.weight)
```

```
# fc2, fc3 layer에 대하여 xavier normal initialization 설정
for name, child in model.named_children():
    for param in child.parameters():
        if name in ['fc2', 'fc3']: # 원하는 layer 이름 지정
            nn.init.xavier_normal_(child.weight)
```

SOLUTIONS FOR MODELING

◆ Gradient Vanishing & Exploding

▪ 해결방법 - **Weight initialization : 헤(He)**

- uniform, normal 분포로 초기화

He Uniform Initialization

$$W \sim U(-\sqrt{\frac{6}{n_{in}}}, \sqrt{\frac{6}{n_{in}}})$$

He Normal Initialization

$$W \sim N(0, Var),\ Var = \sqrt{\frac{2}{n_{in}}}$$

SOLUTIONS FOR MODELING

◆ Gradient Vanishing & Exploding

▪ 해결방법 - **Weight initialization : 헤(He)**

```
# fc1 layer에 대한 He uniform initialization
nn.init.kaiming_uniform_(model.fc1.weight)

# fc1 layer에 대한 He normal initialization
nn.init.kaiming_normal_(model.fc1.weight)
```

SOLUTIONS FOR MODELING

◆ Gradient Vanishing & Exploding

▪ 해결방법 - **Weight initialization : 헤(He)**

```
# fc1 layer에 대한 He uniform initialization
nn.init.kaiming_uniform_( model.fc1.weight )

# fc1 layer에 대한 He normal initialization
nn.init.kaiming_normal_( model.fc1.weight )
```