Deep Learning Process

이현재
Department of Industrial Engineering, Pusan National University

Contents

- Background
- Artificial Neural Network (ANN, 인공신경망)
- Deep Neural Network (DNN, 심층신경망)
- Deep Learning Framework
- Activation Function
- Code

Background

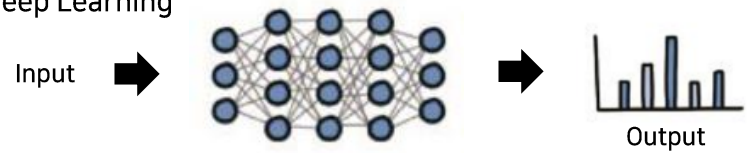
- 인공 지능 (Artificial Intelligence) : 인간의 지능이 갖고 있는 기능을 갖춘 컴퓨터
- 머신 러닝 (Machine Learning) : 컴퓨터가 학습할 수 있도록 하는 알고리즘과 기술을 개발하는 분야
- 딥 러닝 (Deep Learning) : 여러 **비선형** 변환기법의 조합을 통해 높은 수준의 추상화를 시도하는 머신 러닝 알고리즘 집합 ('Deep'은 깊은 통찰이 아닌 **층 기반 학습**을 의미)

Background

- 머신 러닝
  - 샘플과 기댓값이 주어졌을 때 데이터 처리 작업을 위한 실행 규칙을 찾는 것 (입력 데이터에 대한 유용한 변화를 찾는 것)
  - 입력 데이터 포인트 : Input
  - 기대 출력 : Target
  - 알고리즘의 성능을 측정하는 방법 : 알고리즘의 현재 출력과 기대 출력 간의 차이를 결정하기 위해 필요. 측정 값은 알고리즘의 작동 방식을 교정하기 위한 신호로 다시 피드백 되고 이 수정 단계를 **학습**이라고 한다.

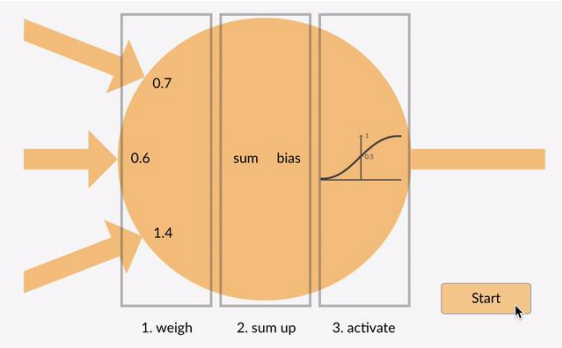
Machine Learning

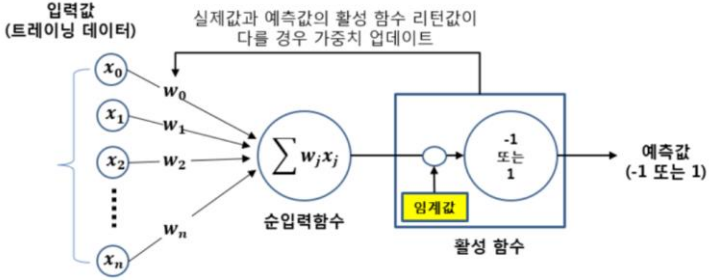
Deep Learning



Artificial Neural Network (ANN, 인공신경망)

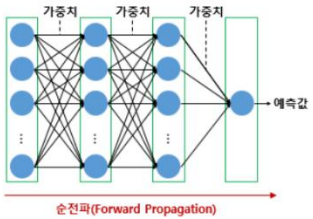
- 사람의 신경망 원리와 구조를 모방하여 만든 기계 학습 알고리즘 (=단층 퍼셉트론)
  - 자극이 어떠한 임계값을 넘어서면 결과 신호를 전달하는 과정에서 착안
- 모든 비선형 함수를 학습
- 모든 Input을 Output에 매핑하여 가중치를 학습할 수 있는 능력
- 구성 : Input, Output, Hidden Layer(활성화 함수를 사용하여 최적의 weight와 bias를 찾아내는 역할)
- 단점
  - 학습 과정에서 파라미터의 최적값을 찾기 어려움 : Gradient Vanishing , Local vs Global
  - 학습 시간이 느림 (과거의 문제, GPU 도입으로 해결)
  - Overfitting (과거의 문제, 충분한 학습데이터 보유)





Deep Neural Network (DNN, 심층신경망)

- 여러 층의 퍼셉트론으로 적어도 2개 이상의 Hidden Layer 보유 (= 다층 퍼셉트론) (Hidden Layer는 비선형 함수를 통과)
- XOR을 극복하기 위해 Hidden Layer를 하나 더 쌓은 DNN이 딥러닝의 시작
- 학습 방법
  - Forward Propagation (순전파) : Input으로부터 정보를 순방향으로 진행하여 예측값 $\hat{y}$을 생성하여 비용 함수 계산
  - Back Propagation (역전파) : 계산된 비용 함수를 이용해 역방향으로부터 정보를 전달하여 가중치 매개변수의 기울기를 효과적으로 계산 (경사 하강법)



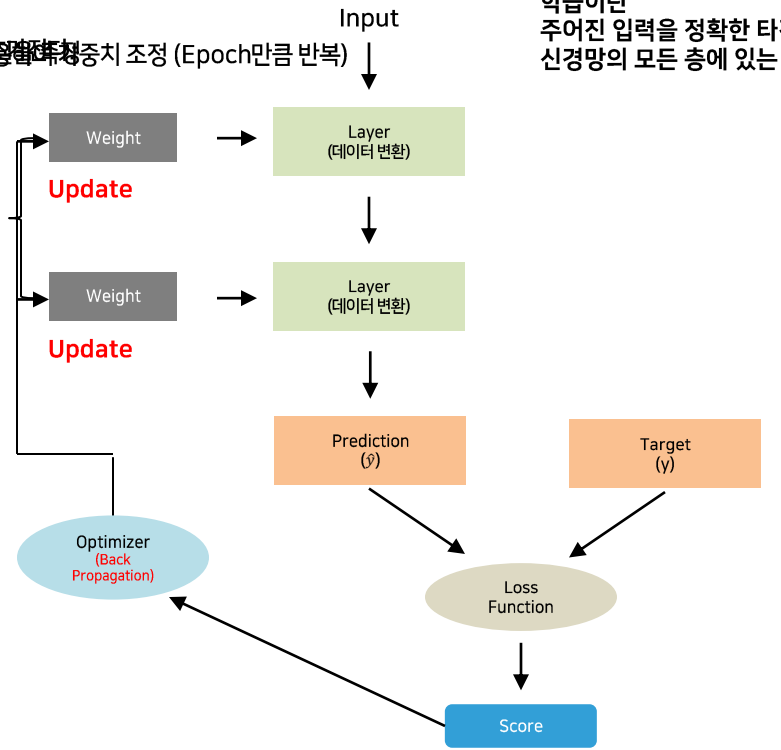
- 한계
  - Gradient Vanishing 여전히 존재

Deep Learning Train Framework

Hidden Layer가 2개 있는 구조

2. 실제 값과 결과값의 오차를 최소화하는 가중치 조정 (Epoch만큼 반복)

목표 : 가중치의 정확한 값을 찾는 것



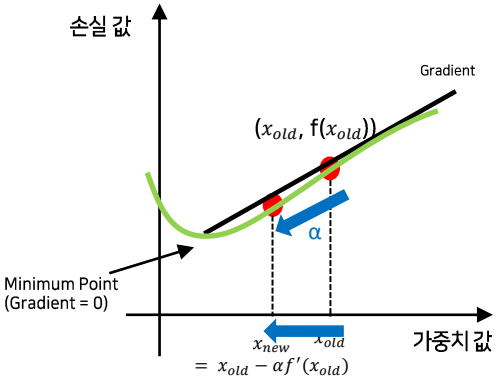
학습이란
주어진 입력을 정확한 타깃에 매핑하기 위해 신경망의 모든 층에 있는 가중치 값을 찾는 것

Gradient descent

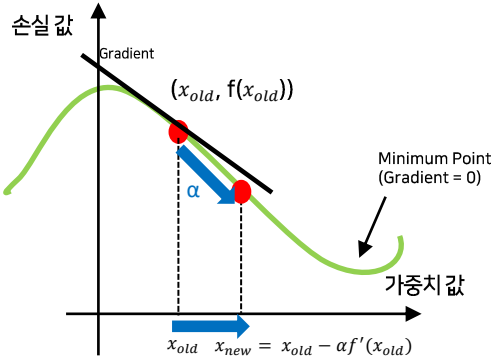
- 접선의 기울기는 증가 함수에는 양의 값, 감소 함수에서는 음의 값을 가진다. (연쇄법칙)

$$x_{new} = x_{old} - \alpha f'(x_{old})$$ α : learning rate

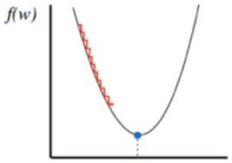
< 기울기가 양수일 때 >



< 기울기가 음수일 때 >

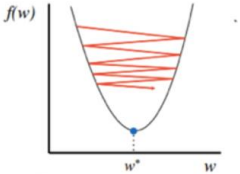


< Learning rate에 따른 경사 하강법 >



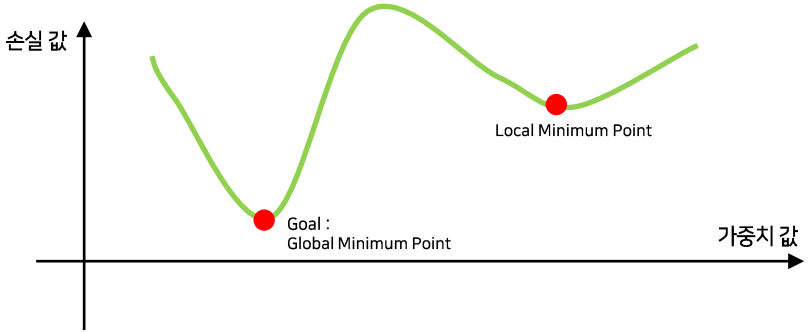
Too small: converge very slowly

or local minimum point에 갇힐 수 있다.



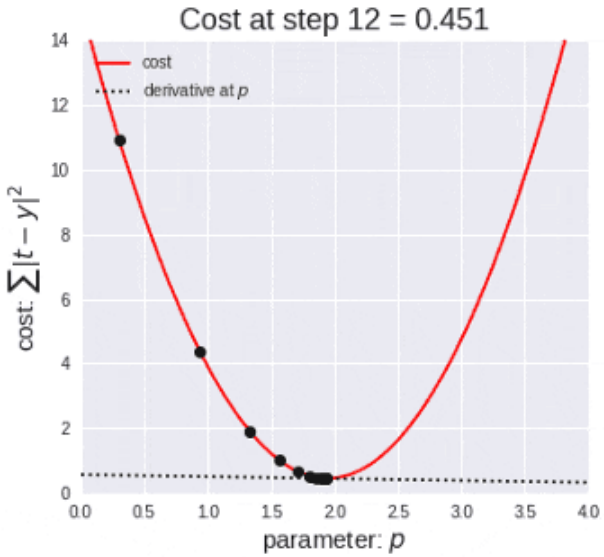
Too big: overshoot and even diverge

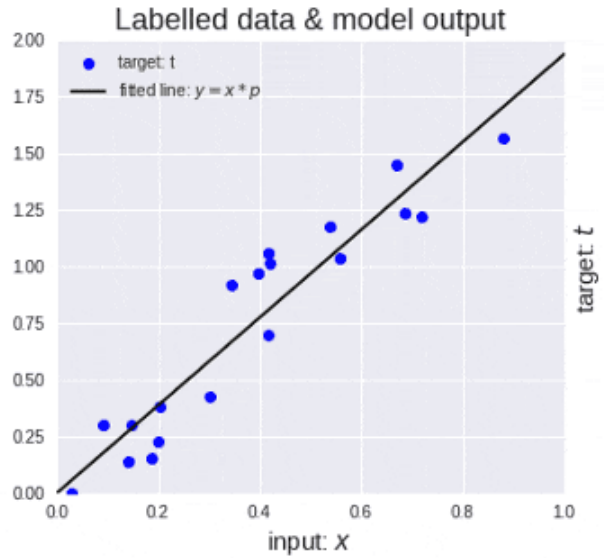
Gradient descent



- 확률적 경사 하강법 (SGD, Stochastic Gradient Descent)
  - 전체 데이터 대신, 일부 데이터(미니 배치)로 업데이트를 진행하는 방식
  - 미니 배치(mini-batch) : 전체 데이터를 N 등분해서 배치 방식으로 처리
  - 볼록이 아닌 비용 함수에도 사용 가능 : Local Minimum Point에 도달해도, 미니 배치를 매번 바꾸면 0이 아니기 때문에 Local Minimum Point에도 멈추지 않고 계속 진행이 가능하다.
  - 일부 데이터만 사용하기 때문에 전체 데이터를 사용했을 때보다 훨씬 빠르지만, 수렴은 비슷하게 한다고 한다.

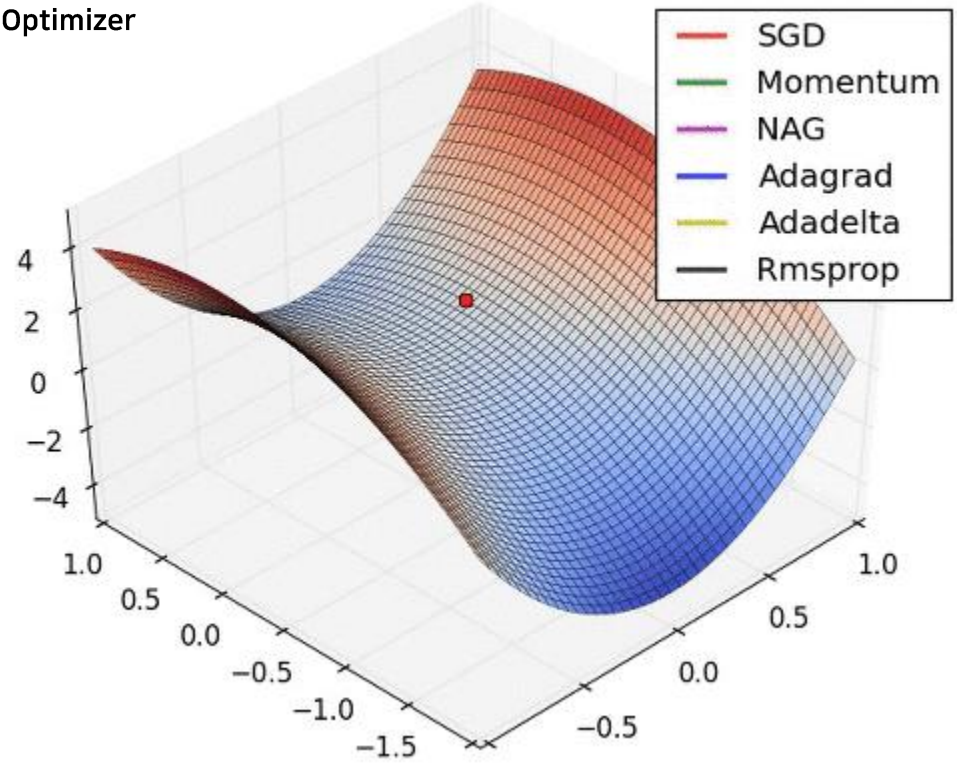
Gradient descent





참고

가중치 초기화, Optimizer



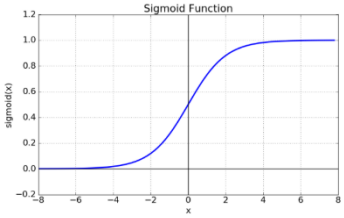
Activation Function

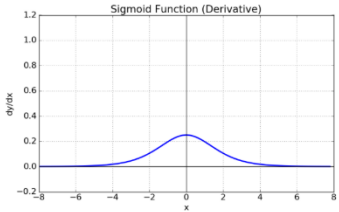
- 비선형 함수를 활성화 함수로 사용해야 하는 이유?
  - 딥러닝 모델의 Layer 층을 더 깊게 쌓기 위해서
  - Input을 다음 Layer로 어떻게 전달할지 결정(활성화할지, 비활성화할지)

- Sigmoid 함수

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

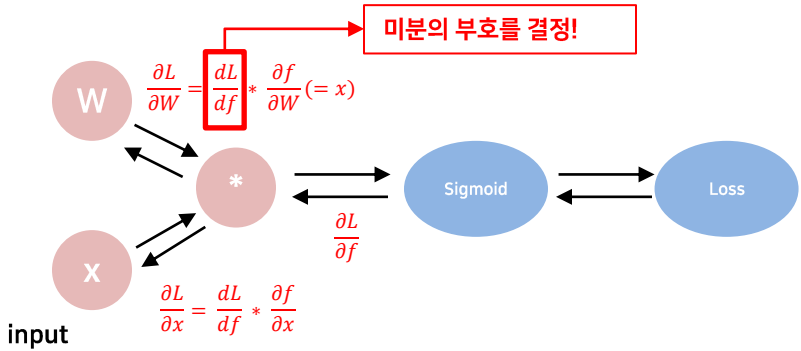
$$\sigma'(x) = \sigma(x)(1-\sigma(x))$$



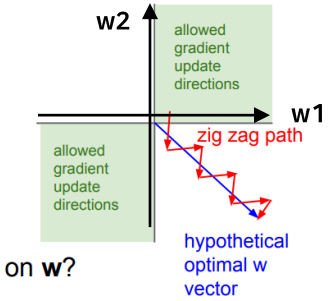


  - Output을 0 ~ 1 사이로 제한(데이터를 섬세하게 분류할 수 있도록 도와주기 때문에 Hidden Layer에서 사용)
  - Gradient Vanishing : 미분 값이 0이 나올 수 있고, 최대값이 0.25이기 때문에 Back Propagation할 때 연쇄법칙에 의해 계속 1보다 작은 수를 곱하게 되면 기울기가 0에 수렴해 학습을 하지 못하는 현상
  - Not zero-centered : 다음 페이지에
  - Exp 연산

Not Zero-Centered



Input이 not zero-centered라서 항상 양의 값이 나온다면?
Gradient of weight의 값은 항상 양수 혹은 음수, Gradient weight는 항상 같은 방향으로 움직인다는 뜻



W의 미분이 항상 양수 혹은 음수이기 때문에
w1이 증가할 때 w2도 증가하거나, w1이 감소할 때 w2이 감소하는 방향으로만 가능

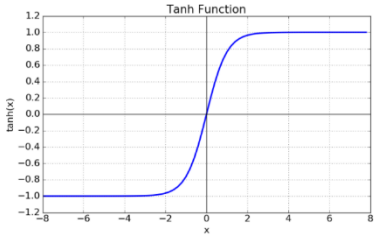
최적해가 w1이 증가했을 때 w2가 감소하는 방향이라면 비효율적으로 탐색

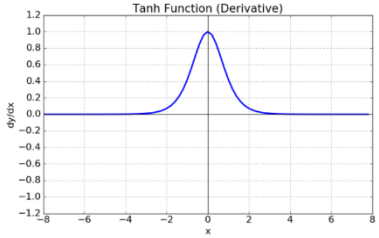
Activation Function

- Tanh 함수

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

$$tanh'(x) = 1 - tanh^2(x)$$





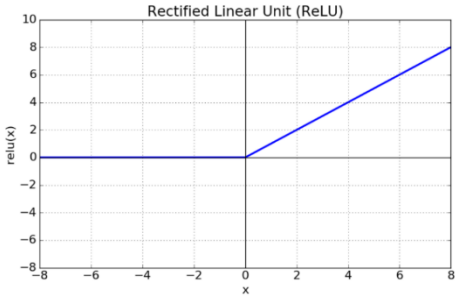
  - Zero-Centered이기 때문에 시그모이드의 최적화 과정이 느려지는 문제를 해결
  - 여전히 Gradient Vanishing 문제 발생

- Gradient Vanishing 때문에 출력층으로 Sigmoid 혹은 Tanh 함수를 사용하지 않는다. 층이 깊어질수록 0에 수렴할 가능성이 높기 때문이다.

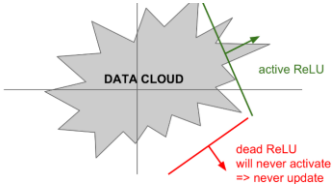
Activation Function

- ReLU 함수

$$f(x) = max(0, x)$$



  - input이 0보다 크면 그 input 그대로 반환하고, 0보다 작으면 0을 반환하는 함수
  - Sigmoid, Tanh 함수보다 학습이 빠르면서 연산 비용이 크지 않고 구현이 매우 간단
  - 은닉층에서 사용하는 것을 추천
  - 양의 값에서는 Gradient Vanishing 문제가 발생하지 않음
  - 한계
    - 음의 값에서는 여전히 Gradient Vanishing 문제 발생 -> Dead ReLU (음수라면 0을 반환하기 때문에 이후로 학습 X)
    - Not Zero-Centered
    - 0에서 미분 불가능 (0에 걸릴 확률이 적으니 무시 가능)

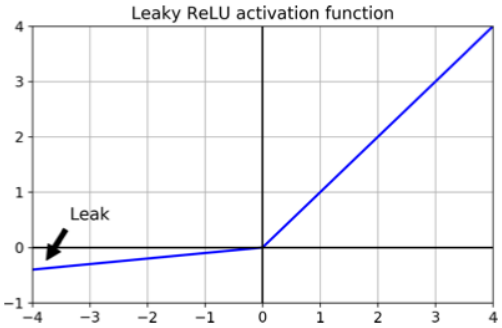


Activation Function

(새는, 구멍이 나는)

- Leaky ReLU

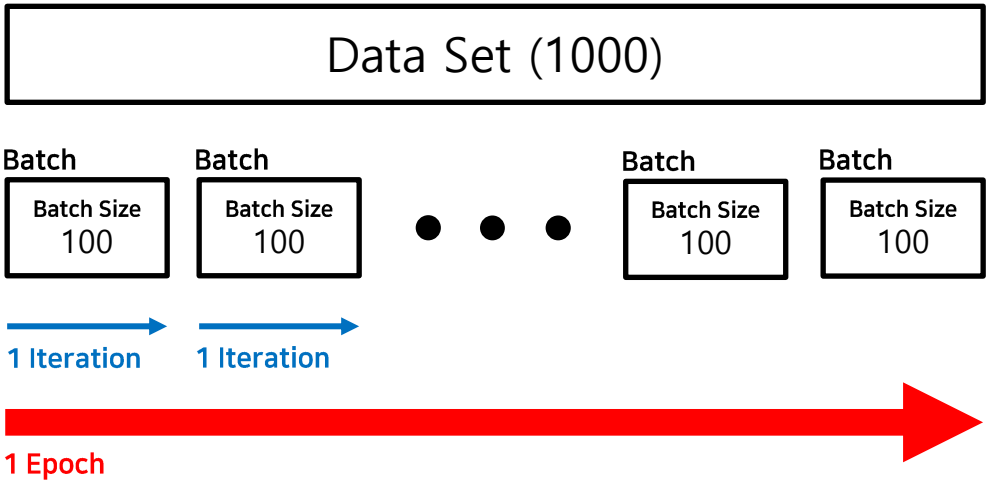
$$f(x) = max(0.01x, x)$$



  - Gradient Vanishing 문제 해결 (음의 값에도 기울기가 0이 아니기 때문에) -> Dead ReLU 해결
  - 여전히 계산이 효율적이며 빠름
  - 여전히 not zero-centered

Epoch, Batch, Iteration

- Epoch : 1 Epoch은 전체 데이터 셋에 대해 forward / backward 즉, 한 번 학습을 한 상태
- Batch (=Mini Batch) : 전체 데이터 셋을 몇 개의 데이터 셋으로 나누었을 때, 하나의 뭉치
- Batch Size : 하나의 배치에 넘겨주는 데이터 개수, 한 번의 배치마다 주는 데이터 Size
- Iteration : 반복, 하나의 미니 배치를 학습할 때



Code

- Model

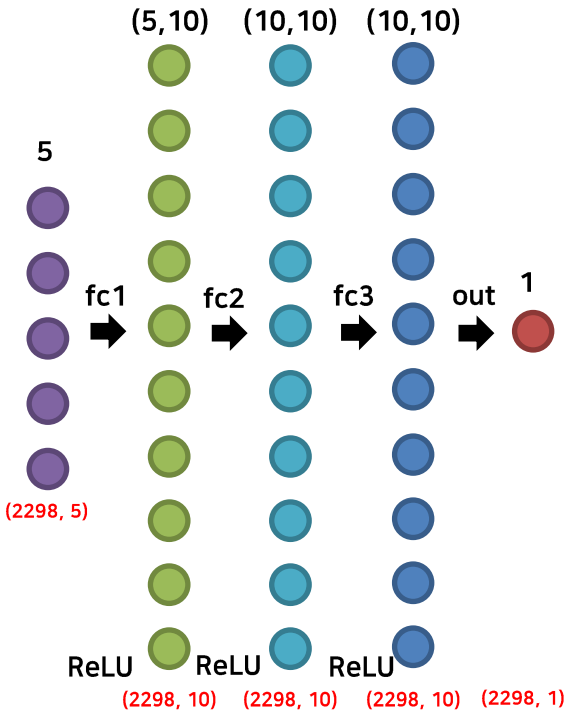
```python
# ANN 모델 생성
class Model(nn.Module):
    # torch.manual_seed(2022)
    # 총 5개의 feature를 input으로 사용해서 output 하나가 나오는 형태, 은닉층은 총 3개, 각 layer 당 10개
    def __init__(self, in_features=5, h1 = 10, h2 = 10, h3 = 10 , out_features=1) :
        super(Model, self).__init__()
        # input -> h1(첫 번째 hidden layer)
        self.fc1 = nn.Linear(in_features, h1)
        # h1 -> h2
        self.fc2 = nn.Linear(h1, h2)
        # h2 -> h3
        self.fc3 = nn.Linear(h2, h3)
        # h3 -> output
        self.out = nn.Linear(h3, out_features)

        # 입력 -> 은닉층 1 -> 은닉층 2 -> 은닉층 3 ->  출력

    #순전파
    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = self.out(x)
        return x
```

```
Device : cuda
모델 생성
첫 번째 Hidden Layer Shape : Linear(in_features=5, out_features=10, bias=True)
두 번째 Hidden Layer Shape : Linear(in_features=10, out_features=10, bias=True)
세 번째 Hidden Layer Shape : Linear(in_features=10, out_features=10, bias=True)
```

```
Input Shape : torch.Size([2298, 5])
Output Shape After First Hidden Layer : torch.Size([2298, 10])
Output Shape After Second Hidden Layer : torch.Size([2298, 10])
Output Shape After Third Hidden Layer : torch.Size([2298, 10])
Output Shape : torch.Size([2298, 1])
```



Code

- Preporcess

```python
from torch.utils.data import TensorDataset # 텐서데이터셋
from torch.utils.data import DataLoader # 데이터로더

# Load Data
data = pd.read_csv('samsung.csv')

X = data.drop(['Date', 'Adj Close'], axis=1)
y = data['Adj Close']


# 데이터 프레임 -> numpy array 형태로 추출
# 학습을 위해서는 DataFrame -> numpy array -> Tensor로 변환
X=X.values
y=y.values

from sklearn.model_selection import train_test_split

# train/test 비율 8 : 2
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=2022)

# 위에서 설명한 데이터 텐서화
X_train = torch.FloatTensor(X_train)
X_test = torch.FloatTensor(X_test)
y_train = torch.FloatTensor(y_train)
y_test = torch.FloatTensor(y_test)

# Train dataset 만들기
dataset = TensorDataset(X_train, y_train)
loader = DataLoader(dataset, batch_size=256, shuffle=False, drop_last= False) # 미니 배치 크기

# 모델 선언
model = Model()

# 손실함수를 MSE로 정의
criterion = torch.nn.MSELoss().to(device)

# 최적화 함수 정의
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
```

```
학습 데이터 X Shape : (2298, 5)
학습 데이터 y Shape : (2298,)
테스트 데이터 X Shape : (575, 5)
테스트 데이터 y Shape : (575,)
```

Code

- Train

```
# 학습

epochs = 100 # 훈련 횟수 100번
loss_list = [] # loss를 담을 리스트

for epoch in range(1, epochs+1):
    loss_sum = 0
    for batch_idx, samples in enumerate(loader):

        X_train, y_train = samples

        model.train()
        y_pred = model(X_train)
        y_pred = y_pred.squeeze()

        loss = criterion(y_pred, y_train)
        loss_sum += loss.item()

        # 역전파 수행
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if epoch % 20 == 0:
            print('Epoch {:4d}/{} Batch {}/{} Cost: {:.0f}'.format(
            epoch, epochs, batch_idx+1, len(loader),
            loss.item()
            ))
    loss_list.append(loss_sum)
```

```
[1 번째 배치 학습] X_train shape : torch.Size([256, 5]), y_train shape : torch.Size([256])
[2 번째 배치 학습] X_train shape : torch.Size([256, 5]), y_train shape : torch.Size([256])
[3 번째 배치 학습] X_train shape : torch.Size([256, 5]), y_train shape : torch.Size([256])
[4 번째 배치 학습] X_train shape : torch.Size([256, 5]), y_train shape : torch.Size([256])
[5 번째 배치 학습] X_train shape : torch.Size([256, 5]), y_train shape : torch.Size([256])
[6 번째 배치 학습] X_train shape : torch.Size([256, 5]), y_train shape : torch.Size([256])
[7 번째 배치 학습] X_train shape : torch.Size([256, 5]), y_train shape : torch.Size([256])
[8 번째 배치 학습] X_train shape : torch.Size([256, 5]), y_train shape : torch.Size([256])
[9 번째 배치 학습] X_train shape : torch.Size([250, 5]), y_train shape : torch.Size([250])
```

Code

- Train

1 Epoch

```
Epoch    1/100 Batch 1/9 Cost: 12019177472
Epoch    1/100 Batch 2/9 Cost: 14006837248
Epoch    1/100 Batch 3/9 Cost: 10556679168
Epoch    1/100 Batch 4/9 Cost: 11010387968
Epoch    1/100 Batch 5/9 Cost: 10698178560
Epoch    1/100 Batch 6/9 Cost: 10770246656
Epoch    1/100 Batch 7/9 Cost: 11579684864
Epoch    1/100 Batch 8/9 Cost: 10755551232
Epoch    1/100 Batch 9/9 Cost: 10995170304
```

20 Epoch

```
Epoch   20/100 Batch 1/9 Cost: 654503232
Epoch   20/100 Batch 2/9 Cost: 594910080
Epoch   20/100 Batch 3/9 Cost: 500231648
Epoch   20/100 Batch 4/9 Cost: 593202048
Epoch   20/100 Batch 5/9 Cost: 562059520
Epoch   20/100 Batch 6/9 Cost: 466732224
Epoch   20/100 Batch 7/9 Cost: 578646592
Epoch   20/100 Batch 8/9 Cost: 508215008
Epoch   20/100 Batch 9/9 Cost: 532902432
```

40 Epoch

```
Epoch   40/100 Batch 1/9 Cost: 615625088
Epoch   40/100 Batch 2/9 Cost: 529427328
Epoch   40/100 Batch 3/9 Cost: 511165856
Epoch   40/100 Batch 4/9 Cost: 562825216
Epoch   40/100 Batch 5/9 Cost: 560246784
Epoch   40/100 Batch 6/9 Cost: 457374496
Epoch   40/100 Batch 7/9 Cost: 568777280
Epoch   40/100 Batch 8/9 Cost: 495732960
Epoch   40/100 Batch 9/9 Cost: 496997760
```

60 Epoch

```
Epoch   60/100 Batch 1/9 Cost: 612272640
Epoch   60/100 Batch 2/9 Cost: 526201024
Epoch   60/100 Batch 3/9 Cost: 509000832
Epoch   60/100 Batch 4/9 Cost: 559789312
Epoch   60/100 Batch 5/9 Cost: 557625344
Epoch   60/100 Batch 6/9 Cost: 455170752
Epoch   60/100 Batch 7/9 Cost: 566006400
Epoch   60/100 Batch 8/9 Cost: 493261856
Epoch   60/100 Batch 9/9 Cost: 494201152
```

80 Epoch

```
Epoch   80/100 Batch 1/9 Cost: 609206144
Epoch   80/100 Batch 2/9 Cost: 523550432
Epoch   80/100 Batch 3/9 Cost: 506507040
Epoch   80/100 Batch 4/9 Cost: 556959616
Epoch   80/100 Batch 5/9 Cost: 554864192
Epoch   80/100 Batch 6/9 Cost: 452919264
Epoch   80/100 Batch 7/9 Cost: 563198016
Epoch   80/100 Batch 8/9 Cost: 490800768
Epoch   80/100 Batch 9/9 Cost: 491692896
```

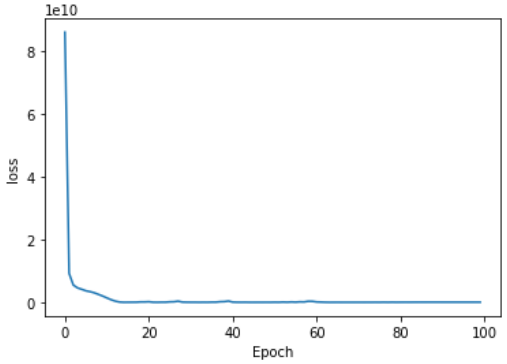
100 Epoch

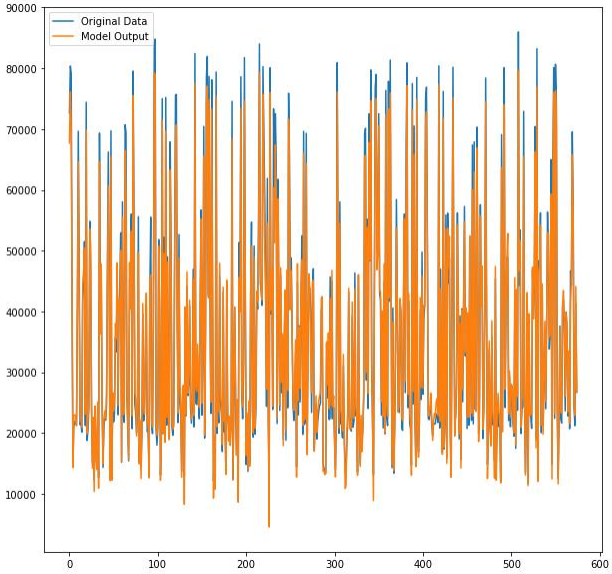
```
Epoch  100/100 Batch 1/9 Cost: 606190208
Epoch  100/100 Batch 2/9 Cost: 520970464
Epoch  100/100 Batch 3/9 Cost: 504052448
Epoch  100/100 Batch 4/9 Cost: 554195072
Epoch  100/100 Batch 5/9 Cost: 552147008
Epoch  100/100 Batch 6/9 Cost: 450701760
Epoch  100/100 Batch 7/9 Cost: 560436416
Epoch  100/100 Batch 8/9 Cost: 488382848
Epoch  100/100 Batch 9/9 Cost: 489252256
```

Code

- Check Test Data

Learning rate = 0.01

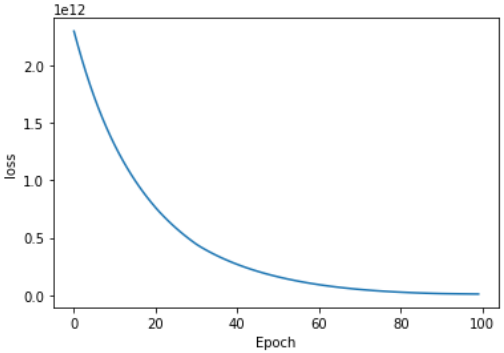


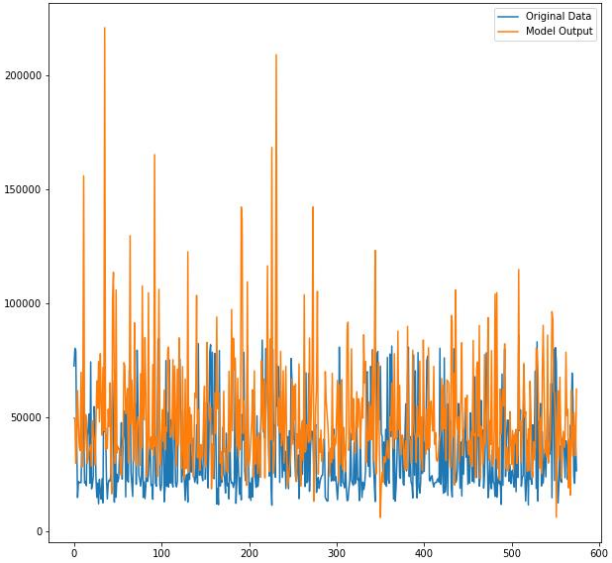


Code

- Check Test Data

Learning rate = 0.0001

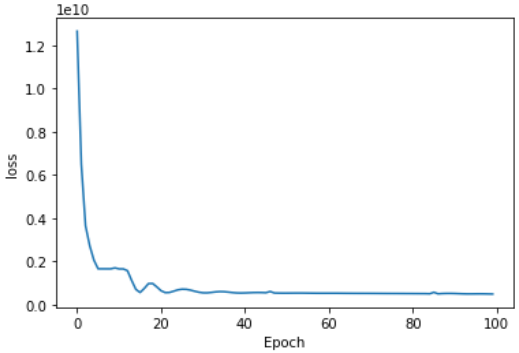


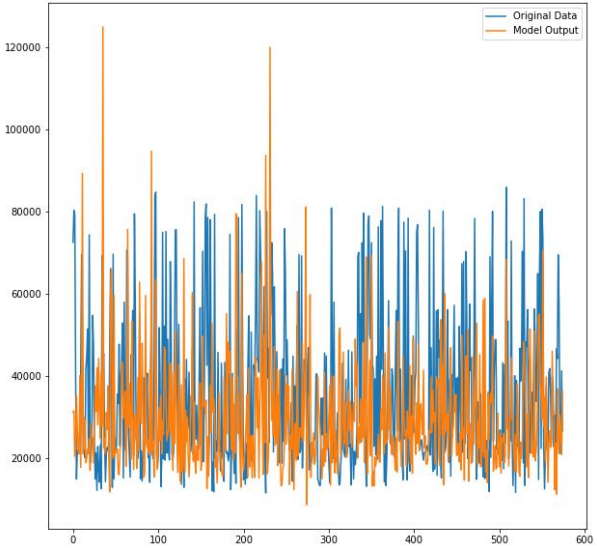


Code

- Check Test Data

No Batch, Learning rate = 0.01





Q&A

감사합니다

이현재