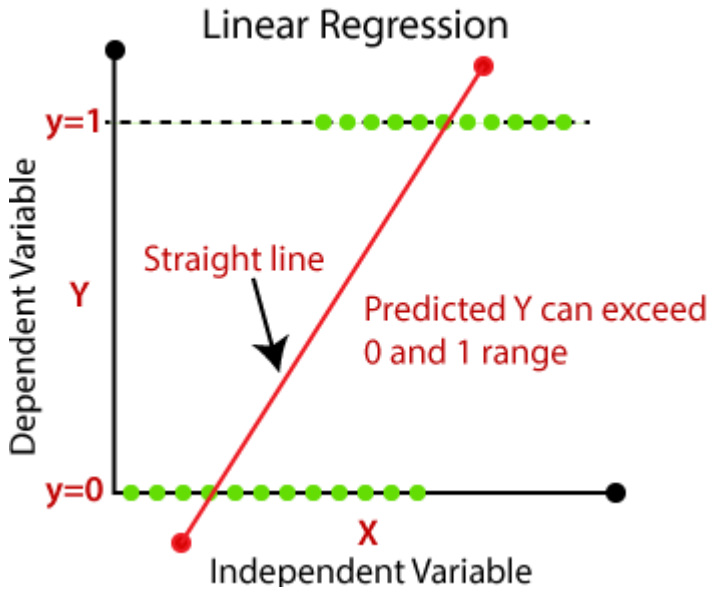
Logistic Regression with PyTorch

2022.01.20

An Introduction to Logistic Regression

- Binary classification을 위한 모델
  - linear regression으로 classification하면 안 될까? 선형 시스템의 한계



1. $H(x) = Wx + b$로 binary distribution을 잘 트래킹할 수 없다.
2. 확률의 정의 $P(Y) = [0,1]$에서 벗어난 값이 나올 수 있다.

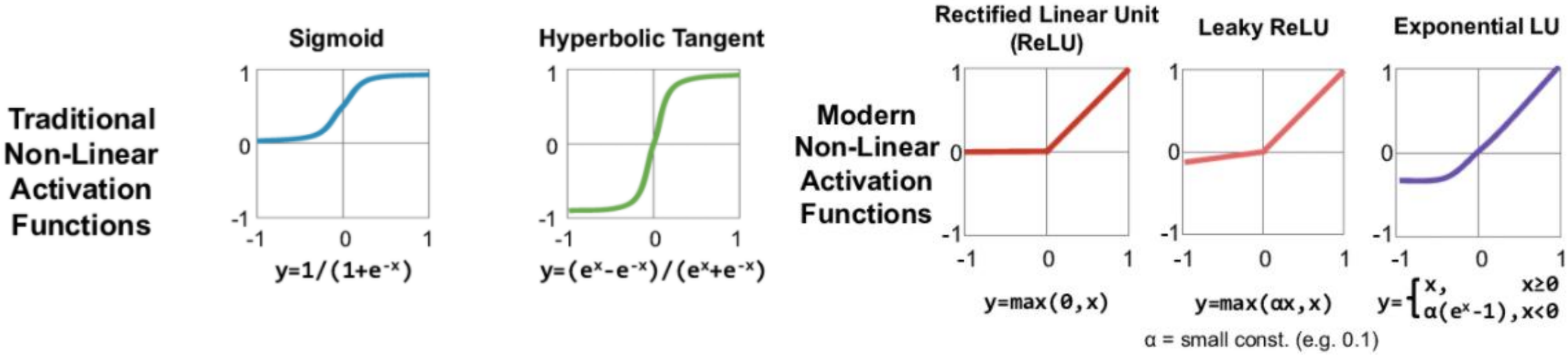
An Introduction to Logistic Regression

- linear regression을 잘 피팅하는 어떤 함수 f를 통과시켜
- $H(X) = f(Wx + b)$으로 수정한다.
- 이때 함수 f로는 시그모이드 함수가 사용된다.

이런 함수 f를 activation function(활성화 함수)라 한다.

Activation function (For Lab09-1)

- 어떤 문제는 선형 시스템으로 해결할 수 없다.
- 선형 시스템을 깊게 쌓으면 되지 않을까?
  - $H(x) = A(A'x + b') + b$ 는 결국 $H(X) = A''x + b''$처럼 선형 시스템
- 따라서 활성화 함수로는 반드시 <u>비선형 함수</u>가 사용된다.



Logistic Regression with PyTorch

- Suggested Algorithm

  - 매 에포크(의 미니배치)마다
  - $X \in \mathbb{R}^{m \times d}$인 input에 weight $W \in \mathbb{R}^{d \times 1}$을 곱한 다음(Linear)
  - Sigmoid function f를 통과해 확률값을 리턴(Activation function)
  - Threshold 0.5를 기준으로 $m \times 1$크기의 결과 리턴(Activation function)
  - Target 변수와 비교
  - Gradient descent 방법을 사용해 weight 업데이트

Logistic Regression with PyTorch

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

# For reproducibility
torch.manual_seed(1)

x_data = [[1, 2], [2, 3], [3, 1], [4, 3], [5, 3], [6, 2]]
y_data = [[0], [0], [0], [1], [1], [1]]

x_train = torch.FloatTensor(x_data)
y_train = torch.FloatTensor(y_data)
```

6 x 2
6 x 1

Logistic Regression with PyTorch

```
        nn.Module을 상속받은 BinaryClassifier 클래스
class BinaryClassifier(nn.Module):
    def __init__(self):
재료  super().__init__()  nn.Module에서 initialize된 객체를 사용하고 싶다
준비  self.linear = nn.Linear(2, 1)   6 x 2 → 2 x 1 통과 => 6 x 1
      self.sigmoid = nn.Sigmoid()   Activation function 통과

    def forward(self, x):
        return self.sigmoid(self.linear(x))  x를 input한 결과
```

Logistic Regression with PyTorch

```
optimizer = optim.SGD(model.parameters(), lr=1)
nb_epochs = 100

for epoch in range(nb_epochs + 1):
    # Hypothesis 구현
    hypothesis = model(x_train) # 선형 회귀식이 시그모이드 함수에 대입

    # Cost 계산
    cost = F.binary_cross_entropy(hypothesis, y_train)

    # Optimizer로 hypothesis 개선
    optimizer.zero_grad()
    cost.backward() # 미분을 통한 gradient descent
    optimizer.step()

    # 10번마다 로그 출력
    if epoch % 10 == 0:
        prediction = hypothesis >= torch.FloatTensor([0.5]) # 1이라고 예측하는 것(
        correct_prediction = prediction.float() == y_train # 1이 맞은 예측
        accuracy = correct_prediction.sum().item() / len(correct_prediction)
        print('Epoch : {:4d}/{} Cost : {:.6f} Accuracy : {:2.2f}'.format(
            epoch, nb_epochs, cost.item(), accuracy*100))
```

1. 최적화 방법 정의
2. 에포크 횟수 정의
3. 에포크 횟수만큼 학습
    1. Hypothesis function 구현
    2. Loss 계산
    3. Optimizer로 Hypothesis 개선
        1. Optimizer 0으로 초기화
        2. 미분 → gradient descent
        3. Optimizer 개선
    4. 학습으로 loss 감소하는지 확인