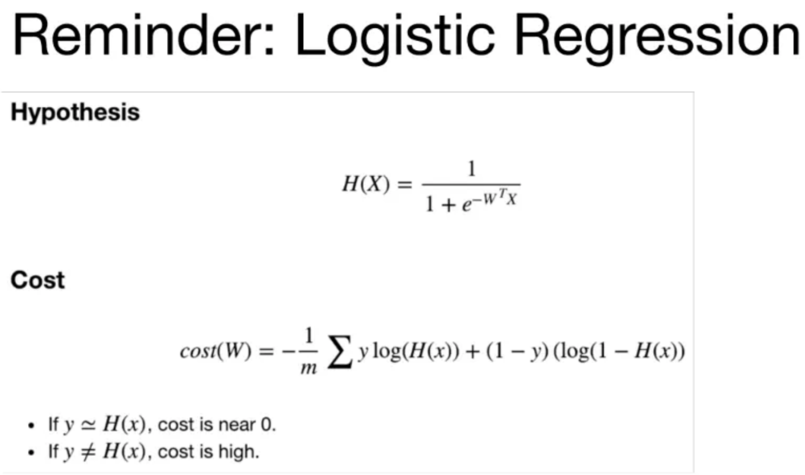
exam	개인학습
Lec	7

Type something...

01) Logistic Regression

01_Theory



Reminder: Logistic Regression

Hypothesis

$$H(X) = \frac{1}{1+e^{-W^T X}}$$

Cost

$$cost(W) = -\frac{1}{m}\sum y\log(H(x)) + (1-y)\,(\log(1-H(x))$$

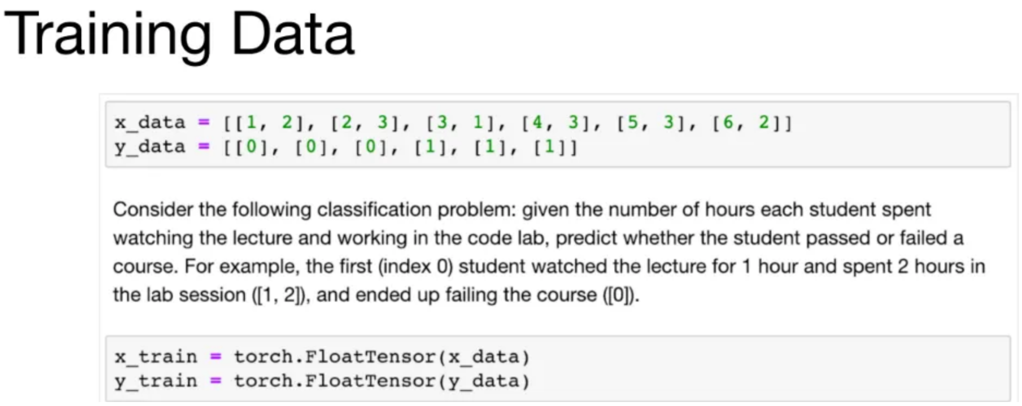
- If $y \simeq H(x)$, cost is near 0.
- If $y \neq H(x)$, cost is high.

$$H(x) = \frac{1}{1+e^{-w^T x}}$$

- Hypothesis는 sigmoid 함수를 사용한다. **Linear + Sigmoid로 생각하면 된다.**

$$cost(W) = \frac{1}{m}\sum ylog(H(x) + (1-y)[log(1-H(x)]$$

- Cost는 x가 0인데 y를 1로 출력하게 되면 -log(h(x)) 가 무한대로 가서 cost가 매우 커지게 된다.

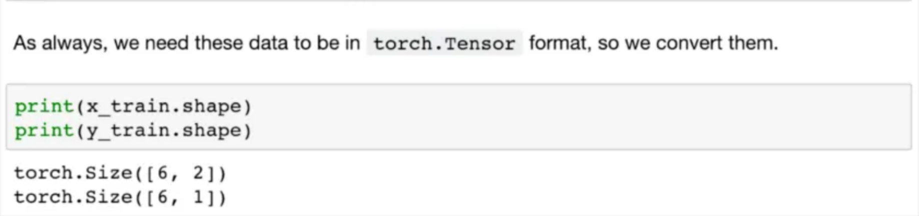


Training Data

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

Consider the following classification problem: given the number of hours each student spent watching the lecture and working in the code lab, predict whether the student passed or failed a course. For example, the first (index 0) student watched the lecture for 1 hour and spent 2 hours in the lab session ([1, 2]), and ended up failing the course ([0]).

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



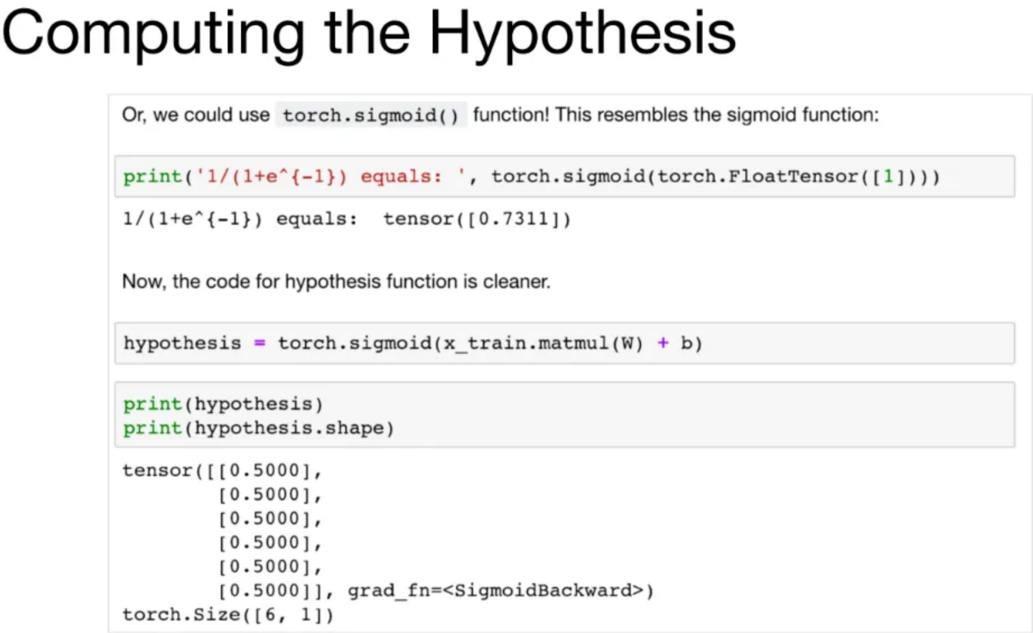
As always, we need these data to be in `torch.Tensor` format, so we convert them.

```
print(x_train.shape)
print(y_train.shape)
```

```
torch.Size([6, 2])
torch.Size([6, 1])
```

- **Logistic Regression은 Binary Classification에 사용된다.**

02_Computing the hypothesis



Computing the Hypothesis

Or, we could use `torch.sigmoid()` function! This resembles the sigmoid function:

```
print('1/(1+e^{-1}) equals: ', torch.sigmoid(torch.FloatTensor([1])))
```

```
1/(1+e^{-1}) equals:  tensor([0.7311])
```

Now, the code for hypothesis function is cleaner.

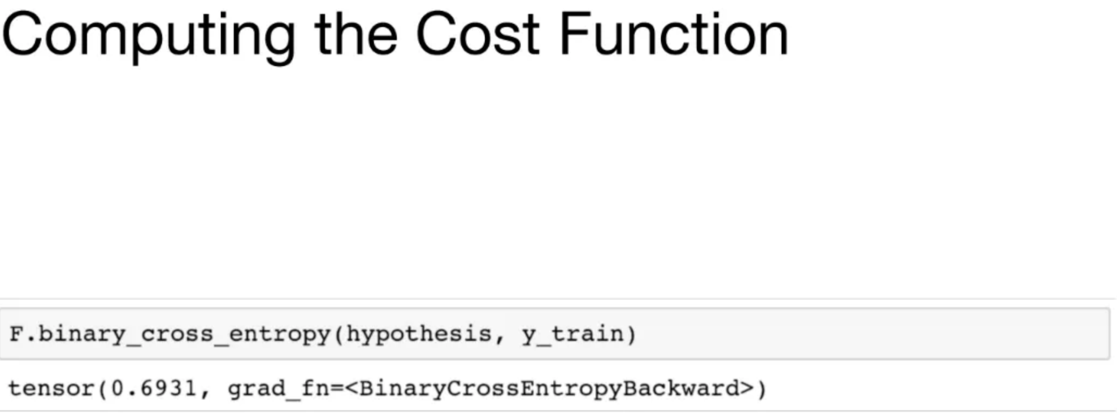
```
hypothesis = torch.sigmoid(x_train.matmul(W) + b)
```

```
print(hypothesis)
print(hypothesis.shape)
```

```
tensor([[0.5000],
        [0.5000],
        [0.5000],
        [0.5000],
        [0.5000],
        [0.5000]], grad_fn=<SigmoidBackward>)
torch.Size([6, 1])
```

- `hypothesis = torch.sigmoid(x_train.matmul(w) + b)` torch.sigmoid를 사용하여 쉽게 계산할 수 있다.

03_Computing the cost function



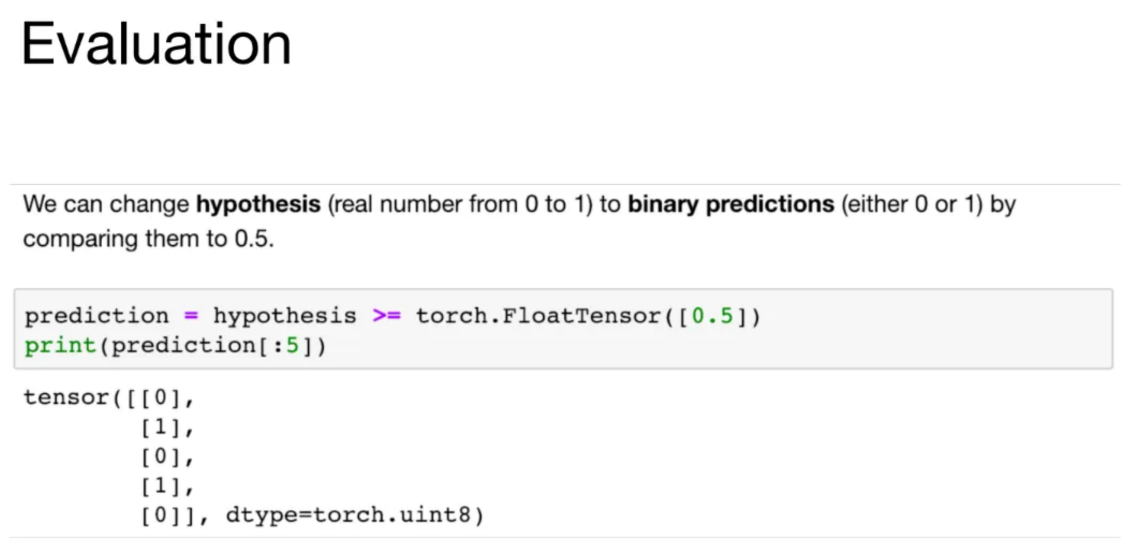
Computing the Cost Function

```
F.binary_cross_entropy(hypothesis, y_train)
```

```
tensor(0.6931, grad_fn=<BinaryCrossEntropyBackward>)
```

- 이진 분류라는 것을 기억해두고, `F.binary_cross_entropy(hypothesis, y_train)` cross_entropy를 사용하자.

04_Evaluation



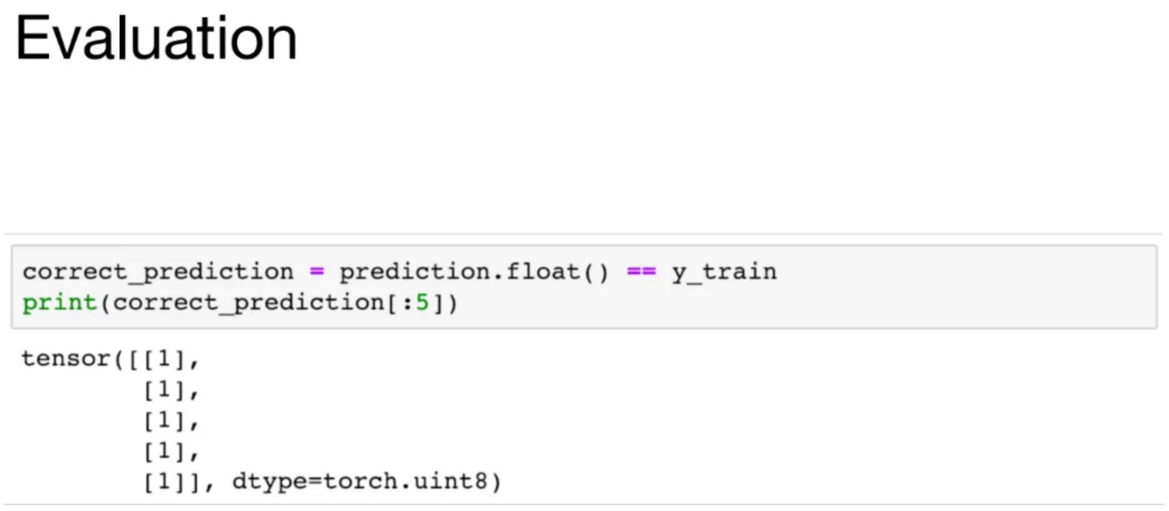
Evaluation

We can change **hypothesis** (real number from 0 to 1) to **binary predictions** (either 0 or 1) by comparing them to 0.5.

```
prediction = hypothesis >= torch.FloatTensor([0.5])
print(prediction[:5])
```

```
tensor([[0],
        [1],
        [0],
        [1],
        [0]], dtype=torch.uint8)
```

- 0.5보다 크면 1로, 0.5보다 작으면 0으로 판정하자. `prediction = hypothesis >= torch.FloatTensor([0.5])`

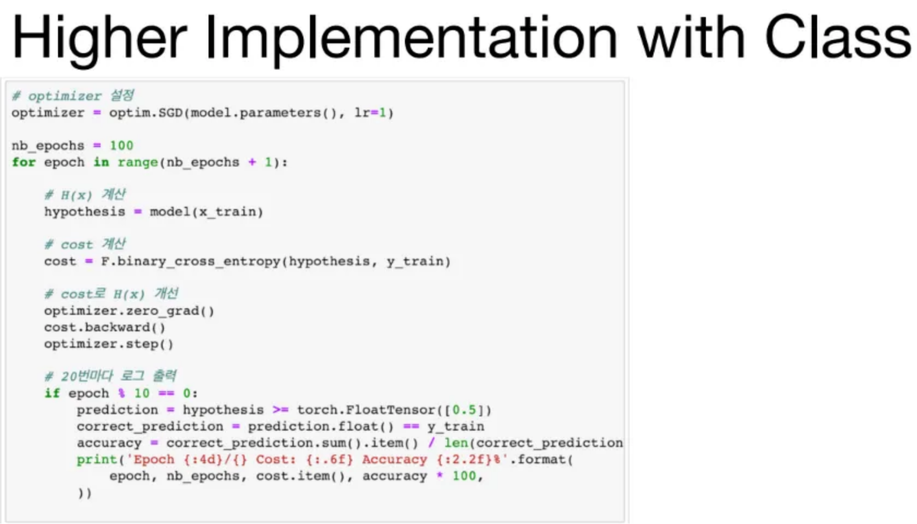


Evaluation

```
correct_prediction = prediction.float() == y_train
print(correct_prediction[:5])
```

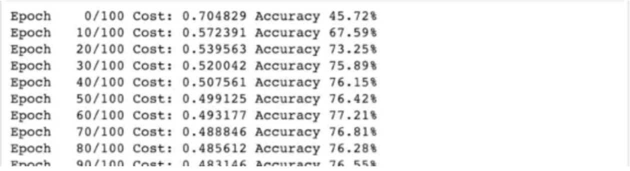
```
tensor([[1],
        [1],
        [1],
        [1],
        [1]], dtype=torch.uint8)
```

- `correct_prediction = prediction.float() == y_train`



Higher Implementation with Class

```
# optimizer 설정
optimizer = optim.SGD(model.parameters(), lr=1)

nb_epochs = 100
for epoch in range(nb_epochs + 1):

    # H(x) 계산
    hypothesis = model(x_train)

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

    # cost로 H(x) 개선
    optimizer.zero_grad()
    cost.backward()
    optimizer.step()

    # 20번마다 로그 출력
    if epoch % 10 == 0:
        prediction = hypothesis >= torch.FloatTensor([0.5])
        correct_prediction = prediction.float() == y_train
        accuracy = correct_prediction.sum().item() / len(correct_prediction
        print('Epoch {:4d}/{} Cost: {:.6f} Accuracy {:2.2f}%'.format(
            epoch, nb_epochs, cost.item(), accuracy * 100,
        ))
```



```
Epoch    0/100 Cost: 0.704829 Accuracy 45.72%
Epoch   10/100 Cost: 0.572391 Accuracy 67.59%
Epoch   20/100 Cost: 0.539563 Accuracy 73.25%
Epoch   30/100 Cost: 0.520042 Accuracy 75.89%
Epoch   40/100 Cost: 0.507561 Accuracy 76.15%
Epoch   50/100 Cost: 0.499125 Accuracy 76.42%
Epoch   60/100 Cost: 0.493177 Accuracy 77.21%
Epoch   70/100 Cost: 0.488846 Accuracy 76.81%
Epoch   80/100 Cost: 0.485612 Accuracy 76.28%
Epoch   90/100 Cost: 0.483146 Accuracy 76.55%
```

- **model를 nn.Module를 상속받은 classifier로 class를 만들 수 있다.**

`hypothesis = model(x_train)`

`cost = F.binary_cross_entropy(hypothesis, y_train)`

`optimzier.zero_grad()`

`cost.backward()`

`optimzier.step()`