<모두를 위한 딥러닝 시즌 2>

김수현

1차시.

Data definition

Hours (x)	Points (y)
1	2
2	4
3	6
4	?

우리가 1,2,3 시간 공부했을 때, 2,4,6점이 나왔다는 사실을 이 표를 통해서 알 수 있다.

이와 같은 데이터를 "Training dataset"이라고 한다.

그리고, 학습이 끝난 후 모델이 얼마나 잘 작동하는지 판별하기 위한 데이터를 "Test dataset"이라고 부른다.

이를 코딩으로 구현해보면, 다음과 같다.

```
x_train = torch.FloatTensor([1], [2], [3])

y_train = torch.FloatTensor([2], [4], [6])
```

여기서 데이터는 torch.tensor 에 해당하고, 입력과 출력을 x_train 과 y_train 으로 구분한다.(즉, x와 y로 구분)

Hypothesis

$$\mathbf{y = Wx + b}$$

Linear regression에서는 학습 데이터와 가장 잘 맞는 하나의 직선을 찾는 일이다. 여기서 기울기 W는 Weight, b는 Bias라고 한다.

Weight와 Bias는 항상 0으로 초기화하는데, 이는 처음에 어떤 입력을 받아도 0을 예측하도록 하

는 것이다. 왜냐하면, W와 b를 학습하도록 하는 것이 목표이기 때문이다.

이를 위해, requires_grad = True 라고 Pytorch에게 명시함으로써, W와 b를 학습시키고 싶다고 알려준다.

Compute loss

학습을 하려면 우선 사용하려고 하는 모델이 얼마나 정답과 가까운지 알아야 하는데, 이 숫자를 cost 또는 loss라고 부르는데 linear regression에서는 Mean Squared Error(MSE)라고 부른다.

이는 예측값과 셋째 Training dataset의 y값의 차이를 제곱해서 평균을 내린 것이라고 할 수 있다.

또한, cost= torch.mean((hypothesis – y_train) ** 2) 로 구할 수 있다.

Gradient descent

Torch.optim 이라는 라이브러리를 사용하는데, [W, b]는 학습할 tensor에 해당하고 lr = 0.01은 learning rate로 지정해준다.

그리고, zero_grad()로 gradient를 초기화하고 backward() 로 gradient를 계산한다. 마지막으로 step()으로 개선을 한다.

Full training code

```
x_train = torch.FloatTensor([1], [2], [3])
y_train = torch.FloatTensor([2], [4], [6]) // 데이터 정의

W = torch.zeros(1, requires_grad = True)
b = torch.zeros(1, requires_grad = True) // Hypothesis 초기화

optimizer = optim.SGD([W,b], lr = 0.01) // Optimizer 정의

nb_epochs = 1000
for epoch in range(1, nb_epochs + 1):
        hypothesis = x_train * W + b // Hypothesis 예측
        cost = torch.mean((hypothesis - y_train) ** 2) // cost 계산

        optimizer.zero_grad() // optimizer로 학습
        cost.backward()
        optimizer.step()
```

2차시.

Simpler Hypothesis Function

위에서 공부했던 $H(x) = Wx + b$ 와는 달리, b라는 bias 벡터 값이 없는 $H(x) = Wx$ 를 구현해볼 수 있다.

Hours (x)	Points (y)
1	1
2	2
3	3

x_train = torch.FloatTensor([1], [2], [3])

y_train = torch.FloatTensor([1], [2], [3])

이 경우에는, $H(x) = x$ 인 경우가 가장 정확한 모델이 될 수 있으므로 W = 1인 경우가 해당한다.

Cost function: intuition

Cost fuction에서는 예측 값과 셋째 데이터가 다르므로 cost가 높아진다. 그래서 cost 함수는 W에 대해 2차 함수와 같이 그려진다.

Cost function을 최소화하기 위해서는 기울기가 음수일때는 W가 더 커져야 되고, 양수일때는 W가 더 작아져야 한다. 또한, 기울기가 가파를수록 cost가 큰 것 이기 때문에 W를 크게 바꾸고, 평평할수록 cost가 평평하게 되니까 W를 작게 바꿔야 한다.

이 기울기를 Gradient Descent 라고 부르는데, 이는 간단한 미분으로 구할 수 있다.

Gradient Descent를 코드로 나타내면 다음과 같다.

gradient = 2 * torch.mean((W * x_train - y_train) * x_train)

lr = 0.1

W -= lr * gradient

Full Code with torch.optim

```
x_train = torch.FloatTensor([1], [2], [3])
y_train = torch.FloatTensor([1], [2], [3]) // 데이터 정의

W = torch.zeros(1, requires_grad = True) // Hypothesis 초기화

optimizer = optim.SGD([W], lr = 0.15) // Optimizer 정의

nb_epochs = 10
for epoch in range(nb_epochs + 1):
        hypothesis = x_train * W  // Hypothesis 예측
        cost = torch.mean((hypothesis - y_train) ** 2) // cost 계산

        print('Epoch {:4d}/{} W: {:.3f} Cost: {:.6f}'.format(
                epoch, nb_epochs, W.item(), cost.item()
        ))
        optimizer.zero_grad() // optimizer로 학습
        cost.backward()
        optimizer.step()
```

위와 같은 코드로 진행하게 되면, W는 점점 1에 수렴하게 되고 cost는 줄어들게 된다.

3차시.

Multivariate Linear Regression

Quiz 1(x1)	Quiz 2(x2)	Quiz 3(x3)	Quiz 4(x4)
73	80	75	152
93	88	93	185
89	91	80	180
96	98	100	196
73	66	70	142

x_train = torch.FloatTensor([[73, 80, 75], [93, 88, 93], [89, 91, 80], [96, 98, 100], [73, 66, 70]])

y_train = torch.FloatTensor([[152], [185], [180], [196], [142]])

Hypothesis Function: Naive 인경우 -> $H(x) = w1x1 + w2x2 + w3x3 + b$ 라고 표현할 수 있는데,

X의 길이가 1000인 vector라면 hypothesis를 계산하는데 어려움이 생길 것이다.

그리고, 복수의 데이터를 다룰 때의 경우를 생각해주는 것이기 때문에 pytorch에서 제공하는 다양한 함수를 적용하여 코드의 간략화 한다.

Full Code with torch.optim

```
x_train = torch.FloatTensor([[73, 80, 75],
                             [93, 88, 93],
                             [89, 91, 80],
                             [96, 98, 100],
                             [73, 66, 70]])
y_train = torch.FloatTensor([[152], [185], [180], [196], [142]])

model = MultivariateLinearRegressionModel()

optimizer = optim.SGD([W, b], lr = le-5) // Optimizer 정의

nb_epochs = 20
for epoch in range(nb_epochs + 1):
        hypothesis = model(x_train) // Hypothesis 예측
        cost = F.mse_loss(prediction, y_train) // cost 계산

        optimizer.zero_grad() // optimizer로 학습
        cost.backward()
        optimizer.step()

        print('Epoch {:4d}/{} hypothesis: {} Cost: {:.6f}'.format(epoch, nb_epochs,
               hypothesis.squeeze().detach(), cost.item()
        ))
```