Tensorflow 강의 2일차 과제

전자공학과 박 상호, 이 진홍

Q1 : What happens to the result if u repeat the program?

-1, 1 사이에서 랜덤 생성된 난수 w, b값을 통해 구해진 선형 모델에 y 값을 빼서 임의의 값 cost 를 생성하여, x_tr, y_tr 의 학습 데이터을 통해 train_tot 만큼 학습하여 최적화한다. 1 회 학습 당 leaning_rate 값만큼 최적화해주며, 결과값 guess는 지정된 learning_rate 와 train_tot 값만큼 최적화된 후 입력 값 test의 출력이다. 입력 값 test 를 5로 하였을 때, 초기 error 값은 5.258, w 값은 0.729, b 값은 -0.459 로, 학습 완료 후 error = 0.001, w = 1.038, b = -0.086으로 결과 값 guess = 5.103 의 결과 값을 출력했다.

Q2 : What is the impact of train_tot?

현재 프로그램에서의 입력 값 test = 5, learning_rate = 0.05 로 설정하고 train_tot 값을 크게 조정 (train_tot = 10000) 하였을 때, 연산 량의 증가로 인해 수행 시간이 증가하나 학습 횟수가 늘어남에 따라 결과 값 guess (5.000)가 유도한 선형 모델에 더욱 근접하는 것을 확인할 수 있었다. train_tot 값에 의해 학습 당 learning_rate 만큼 보정되므로 원하는 선형 모델에 충분히 근접한 결과 값을 얻을 수 있고 리소스가 낭비되지 않는 최적의 train_tot 값을 설정하여야 한다.

Q3 : What is the impact of learning_rate?

현재 프로그램에서의 입력 값 test = 5, train_tot = 100 으로 설정하고 leaning_rate 를 0.0005 로 낮추었을 때, train_tot 만큼 학습을 하여도 보정되는 값이 너무 작아 결과 값 guess(2.845) 가 원하는 결과 값에 근접하지 못하고 종료되었다. 또한, learning_rate 값을 0.5 로 높였을 때는 보정되는 값이 너무 커 목표 값에 수렴하지 못하고 trash 값을 생성하며, 결과값 guess 를 Nan 으로 출력한 채로 종료되었다. 그러므로, 설정한 train_tot 값 내에서 목표 선형 모델에 수렴할 수 있도록 learning_rate 를 조절해야 한다.

Q4 : What happens if you drop (or add) more training data?

입력 값 test = 5, train_tot = 1000, leaning_rate 를 0.005 로 설정하였을 때, 학습 데이터 값 x_tr 과 y_tr 값을 조절함에 따라 다양한 결과값을 얻을 수 있었다. 첫번째로

x_tr = [1, 2, 3]
y_tr = [1, 2, 3] 를 초기값으로 두고

x_tr = [1, 2, 3, 4, 5]
y_tr = [1, 2, 3, 4, 5] 으로 변화를 주어 각각 5회 학습시켰을 때,

기존 결과	변화된 결과
test = 5 guess = 5.212	test = 5 guess = 5.192
test = 5 guess = 4.624	test = 5 guess = 4.956
test = 5 guess = 4.602	test = 5 guess = 4.943
test = 5 guess = 5.139	test = 5 guess = 4.930
test = 5 guess = 5.192	test = 5 guess = 5.002

의 값을 얻을 수 있었다. 이를 통해, 학습 데이터를 기존보다 추가하였을 때 목표 선형 모델에 더 근접한 결과 값 guess 를 얻을 수 있음을 확인하였다. 그렇다면 학습 데이터를 기존 보다 대량 추가하였을 때 더 좋은 값을 얻을 수 있을지 확인해 보기로 하였다.

x_tr = [1, 2, 3, 4, 5 ... ,50]
y_tr = [1, 2, 3, 4, 5 ... ,50] 으로 학습을 진행했을 때, 결과값은

test = 5 guess = nan
test = 5 guess = nan
test = 5 guess = nan
test = 5 guess = nan
test = 5 guess = nan

으로 결과 값 guess 는 nan 으로 출력되었다. 초기 값과 현재 값의 데이터 값을 확인 해봤을 때, 초기 학습을 통해 생성된 error 값이 초기 값보다 학습 데이터를 대량으로 추가한 현재 값이 훨씬 크고, 변동이 심하게 발생하고, 발산하는 것을 확인할 수 있었다. 또한, 학습 데이터를 절반으로 줄여 실행 했을 때도 발산하는 것을 확인할 수 있었다. 그렇다면, 데이터 값의 크기를 촘촘하게 하고 학습 데이터 개수는 50개를 유지해보기로 하였다.

x_tr = [0.1, 0.2, 0.3, 0.4, 0.5 ... ,5.0]
y_tr = [0.1, 0.2, 0.3, 0.4, 0.5 ... ,5.0] 으로 학습을 진행했을 때, 결과값은

test = 5 guess = 4.983
test = 5 guess = 5.015
test = 5 guess = 4.989
test = 5 guess = 5.014
test = 5 guess = 5.024

을 얻을 수 있었는데, 이 결과들을 통해 추론했을 때, 마치 어떠한 DC 신호를 측정할 때, 오실로스코프의 Time Division 과 Volt Division 의 간격을 조정하면 보이는 노이즈 파형과 비슷한 것으로 학습데이터 크기의 간격이 넓었을 때 생성되는 그래프가 학습 데이터의 크기의 간격이 좁을 때 생성되는 그래프보다 더욱 큰 노이즈가 발생하는 것으로 이 노이즈가 보정할 수 없을 만큼 커져

발산하고, 학습데이터 크기의 간격이 좁을 때는 이러한 노이즈의 크기가 작아 발산하지 않고 데이터를 보정하는데 큰 도움을 주는 것으로 예측된다. 이제 반대로 학습 데이터의 수를 줄여보자.

x_tr = [1]
y_tr = [1]

극단적으로 학습 데이터를 하나로 줄여보았다. 이때의 결과값은

test = 5 guess = 3.292
test = 5 guess = 0.861
test = 5 guess = 2.167
test = 5 guess = 3.540
test = 5 guess = 2.244

으로, 목표 선형 모델에서 크게 벗어나는 것을 확인할 수 있었다. 2 번째로 목표 값에 근접하게 학습 데이터를 지정했을 때는

x_tr = [5]
y_tr = [5]

test = 5 guess = 5.000
test = 5 guess = 5.000
test = 5 guess = 5.000
test = 5 guess = 5.000
test = 5 guess = 5.000

x_tr = [4.90]
y_tr = [4.90]

test = 5 guess = 5.007
test = 5 guess = 4.990
test = 5 guess = 4.982
test = 5 guess = 4.978
test = 5 guess = 4.977

결과값 guess 가 학습 데이터가 부족함에도 불구하고 목표 값에 매우 근접하는 것을 확인할 수 있었다. 이 모두를 통해 학습 데이터 값을 설정할 때는 데이터는 많으면 많을수록 좋으나, 너무 급격한 데이터의 변동은 최소화하고 원하는 목표 값을 가능한 수준에서 명확히 설정하는 것이 중요하다는 것을 알 수 있었다.

Q5 : What happens if your test data is very big or small?

입력 값 test = 5, train_tot = 1000, leaning_rate = 0.005, 학습 데이터 간격은 0.1, 테스터 데이터를 각각 3개 씩

x_tr = [499.8, 499.9, 500.0]
y_tr = [499.8, 499.9, 500.0]

x_tr = [0.01, 0.02, 0.03]
y_tr = [0.01, 0.02, 0.03] 으로 설정해 보기로 한다. 각각 실험 결과는

test = 5 guess = nan
test = 5 guess = nan
test = 5 guess = nan
test = 5 guess = nan
test = 5 guess = nan

test = 5 guess = 1.446
test = 5 guess = -0.767
test = 5 guess = -4.253
test = 5 guess = -2.291
test = 5 guess = -4.587

으로, 첫번째 설정은 학습 데이터 값이 너무 커 에러 값이 발산하는 것을 확인할 수 있었고, 두번째 설정은 일정한 값을 만들어내지 못하는 것을 확인할 수 있었다. 앞선 실험 결과와 고려하였을 때, 학습 데이터 값을 설정할 때는 데이터는 많으면 많을수록 좋으나, 너무 급격한 데이터의 변동은 최소화하고, 원하는 목표 값을 가능한 수준에서 명확히 설정하는 것뿐만 아니라, 원하는 목표 값에 크게 벗어난 데이터는 최소화하여 목표 값에 근접한 데이터를 학습 시키는 것이 좋을 것으로 판단된다.

Q6 : What happens if there is a outlier in the train data?

입력 값 test = 5, train_tot = 1000, leaning_rate 를 0.005 로 설정하고,

x_tr = [1, 2, 3, 4, 5, 10]
y_tr = [1, 2, 3, 4, 5, 0.1]

x_tr = [1, 2, 3, 4, 5, 10]
y_tr = [1, 2, 3, 4, 5, 9]

으로 두 가지 학습 데이터를 실행해보았다. 그 결과,

test = 5 guess = 2.374
test = 5 guess = 2.385
test = 5 guess = 2.383
test = 5 guess = 2.379
test = 5 guess = 2.386

test = 5 guess = 4.732
test = 5 guess = 4.731
test = 5 guess = 4.730
test = 5 guess = 4.740
test = 5 guess = 4.740

의도한 선형 그래프에서 벗어난 큰 오차 값의 데이터를 넣었을 때, 목표 선형 모델에서 더욱 멀어지는 것을 확인할 수 있었다.

Q7 : The following shows the number of police officers on patrol vs.
# of crimes reported in the city of Duluth in the last days.

선형 회귀 모델에

x_tr = [10, 15, 16, 1, 4, 6, 18, 12, 14, 7] > 경찰관의 수
y_tr = [5, 2, 1, 9, 7, 8, 1, 5, 3, 6] > 범죄 발생

의 학습 데이터를 넣고, learning_rate = 0.005, train_tot = 10000 회, 경찰의 수 test 를 30으로 하였을 때, y = -0.493 * x + 9.780 의 선형 그래프를 얻을 수 있었고, 결과 값은 -5.01 값이 나오며, 해당 학습 데이터를 사용하여 얻어낸 그래프를 이용한다면 범죄는 일어나지 않을 것이라 예측할 수 있다.

Perceptron---------- 15

Q1 : Modify and2.py to perform 2-input OR function

- 첨부 파일 참조 (OR2.py)

Q2 : Modify and2.py to perform 3-input AND function

- 첨부 파일 참조 (AND3.py)

Q3 : Modify the code to use gradient descent

- 첨부 파일 참조 (AND2GR.py)

해당 소스를 작성할 때, step 함수를 사용하지 못하였는데, 이는 Gradient Descent 사용 시 알 수 없는 이유로 minimize 가 안되는 현상이 발생하였는데 이는 Step 함수가 선형이 아니기 때문에 또는 작성자들의 실력 부족으로 인한 알고리즘 이해 부족이 원인으로 예측된다. 여러 방식을 테스트 해봤을 때, Gradient Descent step 함수가 아니라 tanh 함수를 적용하자 gradient descent 의 동작을 확인할 수 있었다.

---------- 22

Q1 : What is the impact of learning rate in Gradient descent training?

leaning rate 값은 경사 하강법에서 계산된 slope 값이 최소가 아닐 때, 매 학습마다 w 와 b 의 값을 보정하기위해 설정해주는 값이다. Learning rate를 설정할 때는 learning rate 값을 너무 크게 하면 목표 선형 모델에 수렴하지 못하고, 너무 작게 하면 많은 학습 횟수를 동반하여 리소스 낭비가 발생하므로 적정 수준의 조절이 필요하다.

Q2 : What happens if we use step function from AND2, instead of tanh, for activation?

AND2 의 step function 을 tanh, 즉, 쌍곡선 함수 tanh 로 변환하였을 때,

```
def step(x):
    is_greater = tf.greater(x, 0)
    as_float = tf.to_float(is_greater)
    doubled = tf.multiply(as_float, 2)
    return tf.subtract(doubled, 1)

output = step(tf.matmul(train_in, w))
error = tf.subtract(train_out, output)
mse = tf.reduce_mean(tf.square(error))
```

에서

```
    output = tf.tanh(tf.matmul(train_in, w))
    error = tf.subtract(train_out, output)
    mse = tf.reduce_mean(tf.square(error))

    delta = tf.matmul(train_in, error, transpose_a=True)
    train = tf.assign(w, tf.add(w, delta))
```

로 바뀌게 된다. 연산 된 결과 값을 출력하였을 때,

Weight/bias	Weight/bias
[[4.42532396]	[[5.30106258]
[4.04470015]	[5.30106258]
[-3.44273186]]	[-5.30106258]]

Output	Output
[[1.]	[[0.99995029]
[-1.]	[-0.99995029]
[-1.]	[-0.99995029]
[-1.]]	[-1.]]

으로 step function 의 epoch 는 3 회로 매우 적고, tanh function 의 epoch 는 10000 회가 수행되었음에도 불구하고, 약간의 오차를 확인할 수 있었다.

Q3 : Why is T and F float instead of Boolean?

Boolean 체계에서 Ture = 1, False = 0 이다. 그러나, 학습 실행 시 목표 타겟으로 error 값을 보정할 때 실수 계산이 필요하므로 T, F 를 float 변환하여 사용한다.