3장. 불량 철판 판별 분류 신경망

부산대학교 정보컴퓨터공학부

15학번 이연걸

1. 개요

이번 과제는 불량 철판을 선택 분류를 하는 신경망 모델을 만드는 것이다. 데이터셋은 이탈리아 철강 회사 제공 자료로 1941항목으로 구성되어 있으며 34필드로 27필드는 육안으로 확인 가능한 철판들의 각종 특징값이고 나머지 7필드는 원-핫 벡터 형태의 불량 유형 정보이다. (Pastry, Z_Scratch, K_Scratch, Stains, Dirtines, Bumps, Other_Faults)

이런 선택 분류 문제에서는 이진 분류에서 사용했던 시그모이드 함수와 MSE는 다르게 로짓값 벡터를 확률 분포로 변환하기 위해서 소프트맥스 함수를 사용하고 교차 엔트로피를 사용하게 되는데, 엔트로피를 통해 신경망 추정 확률 분포와 정답 확률 분포간의 손실 값을 계산하게 된다.

따라서 왜 선택 분류에서는 시그모이드와 MSE보다는 소프트맥스와 교차 엔트로피를 사용하는지 각각 실행해보고 결과를 분석해본다.

2. 서론

2.1 Softmax 함수란?

Softmax는 입력 받은 값을 출력으로 0~1사이의 값으로 모두 정규화 하며 출력 값들의 총 합은 항상 1이 되는 특성을 가진 함수다. 분류하고 싶은 클래스 수만큼 출력으로 구성한다. 가장 큰 출력 값을 부여받은 클래스가 확률이 가장 높은 것으로 이용된다.

2.2 Cross Entropy란?

딥러닝에서는 분류 문제에 대한 Cost Function으로 Cross Entropy를 사용한다. 분류의 대상이 참, 거짓처럼 2개인 경우 binary cross entropy를 사용하고 이미지넷과 같이 수많은 종류의 대상을 분류하는 경우에는 multi cross entropy를 사용한다.

2.3 Sigmoid 함수

Sigmoid 함수는 주로 로지스틱 회귀분석 또는 Nearal networkdml Binary classification 마지막 레이어의 화렁함수로 사용된다.

2.4 Mean Squared Error

MSE 는 평균제곱오차로서 오차의 제곱에 대해 평균을 취한 것이다. 따라서 항상 0 이상의 수가 결과로 나오며 값이 낮을수록 원본과의 오차가 적다.

2.5 Cross Entropy Loss Function 을 이용한 학습에 대한 분석

Cross Entropy 는 두 개의 확률분포 P 와 q 에 대해 하나의 사건 X 가 갖는 정보량으로 정의된다. 즉, 서로 다른 두 확률분포에 대해 같은 사건이 가지는 정보량을 계산한 것이다. 이는 q 에 대한 정보량을 p 에 대해서 평균낸 것을 볼 수 있다. 식은 다음과 같다.

$$H_{p,q}(X) = -\sum_{i=1}^{N} p(x_i) \log q(x_i)$$

Cross entropy 는 기계학습에서 손실함수(loss function)을 정의하는데 사용되는데, p 는 true probability 로써 true label 에 대한 분포를, q 는 현재 예측모델의 추정값에 대한 분포를 나타낸다.

3. 학습결과

3.1 소프트맥스와 교차 엔트로피 사용

우선, 다음과 같이 정의된 소프트맥스와 교차 엔트로피 함수를 통해 학습을 진행하였다.

```
def softmax(x):
    max_elem = np.max(x, axis=1)
    diff = (x.transpose() - max_elem).transpose()
    exp = np.exp(diff)
    sum_exp = np.sum(exp, axis=1)
    probs = (exp.transpose() / sum_exp).transpose()
    return probs

def softmax_derv(x, y):
    mb_size, nom_size = x.shape
    derv = np.ndarray([mb_size, nom_size, nom_size])
    for n in range(mb_size):
        for i in range(nom_size):
            for j in range(nom_size):
                derv[n, i, j] = -y[n,i] * y[n,j]
            derv[n, i, i] += y[n,i]
    return derv

def softmax_cross_entropy_with_logits(labels, logits):
    probs = softmax(logits)
    return -np.sum(labels * np.log(probs+1.0e-10), axis=1)

def softmax_cross_entropy_with_logits_derv(labels, logits):
    return softmax(logits) - labels
```

그 결과는 다음과 같았다.

Table 1 소프트맥스와 교차엔트로피를 사용한 경우 1

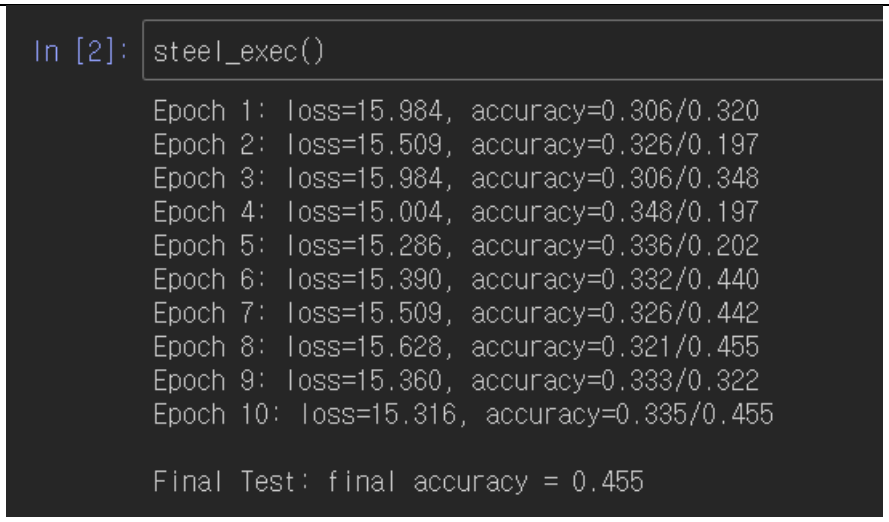
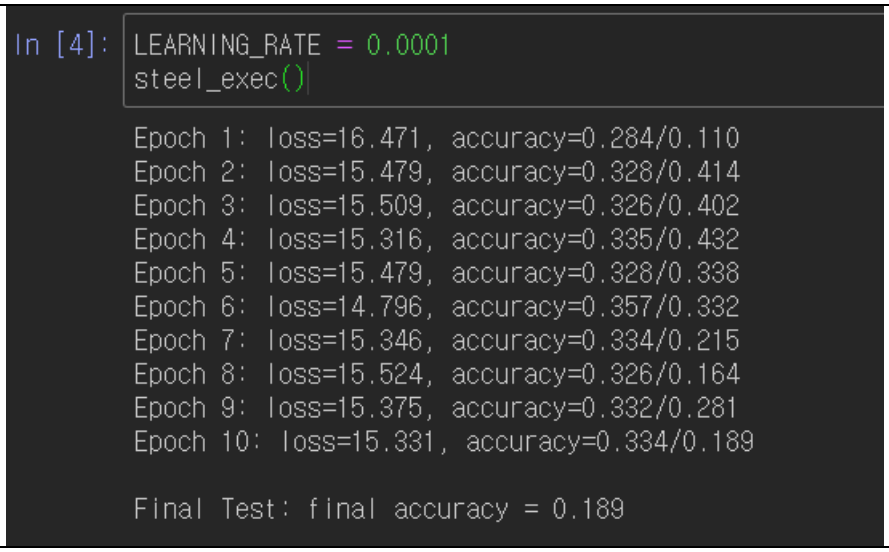
구분	내용
데이터 정확도	45.5%
결과화면	

Table 2 소프트맥스와 교차엔트로피를 사용한 경우 2

(학습률을 0.0001로 조정한 경우)

구분	내용
데이터 정확도	18.9%
결과화면	

3.2 시그모이드와 MSE 사용

그 뒤, 다음과 같이 정의를 한 시그모이드와 MSE 함수를 통해 학습을 진행하였다.

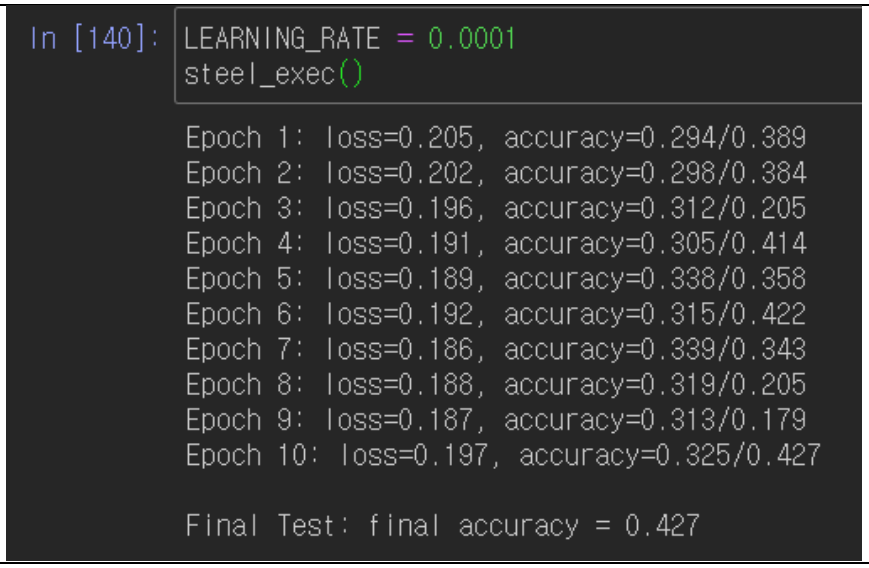
```
In [3]: def forward_postproc(output, y):
            diff = sigmoid(output) - y
            square = np.square(diff)
            loss = np.mean(square)

            return loss, diff

        def backprop_postproc(G_loss, diff):

            shape = diff.shape

            g_loss_square = np.ones(shape) / np.prod(shape)
            g_square_diff = 2 * diff
            g_diff_output = 1

            G_square = g_loss_square * G_loss
            G_diff = g_square_diff * G_square
            G_output = g_diff_output * G_diff

            return G_output
```

Table 3 시그모이드와 MSE를 사용한 경우 1

구분	내용
데이터 정확도	50.1%
결과화면	In [138]: steel_exec() Epoch 1: loss=0.201, accuracy=0.302/0.312 Epoch 2: loss=0.201, accuracy=0.310/0.419 Epoch 3: loss=0.194, accuracy=0.323/0.496 Epoch 4: loss=0.192, accuracy=0.333/0.437 Epoch 5: loss=0.200, accuracy=0.309/0.419 Epoch 6: loss=0.189, accuracy=0.325/0.192 Epoch 7: loss=0.193, accuracy=0.332/0.284 Epoch 8: loss=0.195, accuracy=0.299/0.501 Epoch 9: loss=0.194, accuracy=0.314/0.465 Epoch 10: loss=0.191, accuracy=0.322/0.501 Final Test: final accuracy = 0.501

Table 4 시그모이드와 MSE를 사용한 경우 2

(학습률을 0.0001로 조정한 경우)

구분	내용
데이터 정확도	42.7%
결과화면	

```
In [140]: LEARNING_RATE = 0.0001
          steel_exec()

          Epoch 1: loss=0.205, accuracy=0.294/0.389
          Epoch 2: loss=0.202, accuracy=0.298/0.384
          Epoch 3: loss=0.196, accuracy=0.312/0.205
          Epoch 4: loss=0.191, accuracy=0.305/0.414
          Epoch 5: loss=0.189, accuracy=0.338/0.358
          Epoch 6: loss=0.192, accuracy=0.315/0.422
          Epoch 7: loss=0.186, accuracy=0.339/0.343
          Epoch 8: loss=0.188, accuracy=0.319/0.205
          Epoch 9: loss=0.187, accuracy=0.313/0.179
          Epoch 10: loss=0.197, accuracy=0.325/0.427

          Final Test: final accuracy = 0.427
```

4. 결론

소프트맥스 함수를 사용한 실험과 시그모이드를 사용한 실험 모두 결과에 큰 차이는 없었다. 반복씩 실행해본 결과, 20%에서 40% 수준을 계속 왔다갔다했기 때문이다. 학습률을 변화시켜도 큰 의미는 없었다. 이는 데이터 수가 매우 적기 때문이다. 27차원 벡터 공간을 7개 구역으로 적절히 분할 할 수 있어야 하는데, 1941개에 불과한 데이터셋은 너무 적다.

한편, 소프트맥스와 시그모이드 모두 차이가 없다는 점에서 두 함수는 본질적으로 비슷한 기능을 한다는 점을 느낄 수 있었다. 시그모이드 함수를 적용하면 각 항목이 답으로 선택될 확률이 따로 구해질 뿐 그 합이 1로 맞추어진다는 제약이 사라지고 즉, 확률 분포가 아닌 개별적인 확률들이 구해진다. 따라서 그 중 가장 큰 확률을 보이는 항목을 답으로 선택할 수 있고 소프트맥스 함수를 이용할 때처럼 비슷한 결과를 지니게 되는 것이다.

시그모이드 함수는 소프트맥스의 입력 변수를 하나 줄인 함수에 해당한다. 따라서 7개 후보 항목이 있다고 가정하면 퍼셉트론을 6개만 두고 한 항목에 대한 로짓값은 항상 0으로 간주하여 소프트맥스 함수나 교차 엔트로피 함수를 계산하고 그 편미분을 따져도 된다. 하지만 이렇게 하면 기준이 되는 한 항목을 별도로 취급하기 위해 코드가 쓸데없이 복잡해진다. 따라서 이진 판단에서는 시그모이드를 이용하지만 선택 분류에서는 거의 쓰이지 않는다.