3장. 철판 분류 신경망 과제

부산대학교 전기컴퓨터공학부 정보컴퓨터공학전공
201724579 정현모

1 개요

많은 인공지능 모델에서 Cost Function으로 Mean – Squared – Error를 사용하는 것이 일반적이다. 하지만 classfication모델에선 Cost Function으로 Cross – Entropy함수를 자주 사용한다. 이번엔 왜 classfication모델에서 Cross – Entropy함수가 자주 쓰이는 지 알아보고, Activation Function으로 Sigmoid와 Softmax함수를 사용해 봄으로써 성능을 비교해본다.

2 학습 결과

2.1 모델 1

구분	내용
조건	Softmax + Cross-Entropy 모델
최종 결과	Accuarcy: 45.5%
결과 화면	Epoch 1: loss=15.984, accuracy=0.306/0.320 Epoch 2: loss=15.509, accuracy=0.326/0.197 Epoch 3: loss=15.984, accuracy=0.306/0.348 Epoch 4: loss=15.004, accuracy=0.348/0.197 Epoch 5: loss=15.286, accuracy=0.336/0.202 Epoch 6: loss=15.390, accuracy=0.332/0.440 Epoch 7: loss=15.509, accuracy=0.326/0.442 Epoch 8: loss=15.628, accuracy=0.321/0.455 Epoch 9: loss=15.360, accuracy=0.333/0.322 Epoch 10: loss=15.316, accuracy=0.335/0.455 Final Test: final accuracy = 0.455

[1] https://curt-park.github.io/2018-09-19/loss-cross-entropy/

3장. 철판 분류 신경망 과제

2.2 모델 2

구분	내용
조건	Sigmoid + MSE 모델
최종 결과	Accuarcy: 76.4%
결과 화면	Epoch 1: loss=1.399, accuracy=0.800/0.795 Epoch 2: loss=1.346, accuracy=0.808/0.849 Epoch 3: loss=1.359, accuracy=0.806/0.866 Epoch 4: loss=1.353, accuracy=0.807/0.771 Epoch 5: loss=1.341, accuracy=0.808/0.865 Epoch 6: loss=1.379, accuracy=0.803/0.867 Epoch 7: loss=1.344, accuracy=0.808/0.838 Epoch 8: loss=1.326, accuracy=0.811/0.832 Epoch 9: loss=1.332, accuracy=0.810/0.864 Epoch 10: loss=1.335, accuracy=0.809/0.764 Final Test: final accuracy = 0.764

3 결과 분석

3.1 Mean-Squared-Error vs Cross-Entropy

Mean-Squared-Error(MSE)와 Cross-Entropy의 식은 아래와 같다.

$$J(W,b;x,y) = \frac{1}{2}\frac{1}{n}\sum_{x}(\sigma(z)-y)^2$$

그림 1. MSE 함수

$$J(W,b;x,y) = -\frac{1}{n}\sum_{x}[ylna + (1-y)ln(1-a)]$$

그림 2. Cross-Entropy 함수

식에서 알 수 있듯이 MSE 함수는 실제 값과 예측 값의 차이를 제곱해서 모두 더한 것으로 정의되고, Cross-Entropy 함수는 실제 값과 예측 값의 차이를 극대화하는 함수이다. 함수 그래프에서 알 수 있듯이 정답과 차이가 적을수록 비용이 작아지게 된다.

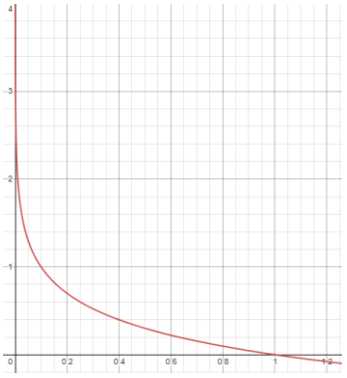

그림 3. Cross-Entropy 함수 그래프

[1] https://curt-park.github.io/2018-09-19/loss-cross-entropy/

3장. 철판 분류 신경망 과제

3.2 Sigmoid vs Softmax

Sigmoid Function과 Softmax Function의 식은 아래와 같다.

$$sigmoid(x_i) = \frac{1}{1 + e^{(-x_i)}}$$

그림 4. Sigmoid Function

$$softmax(x) = \frac{x_i}{\sum_{j=0}^{k} e^{x_j}} \ (i = 0, 1, \ldots k)$$

그림 5. Softmax Function

Sigmoid 함수는 들어오는 x값에 따라 0~1사이의 값을 반환한다. softmax 함수는 들어오는 x값 전체에서 해당 x값이 차지하는 확률을 구한다. 즉, 모든 x값의 합을 1이라 뒀을 때 그 중 x값이 차지하는 영역을 의미한다.

3.3 Sigmoid + MSE vs Softmax + Cross-Entropy

Sigmoid Function과 Mean Squared Error를 조합한 모델과 Softmax + Cross Entropy를 조합한 모델을 비교해보았다. 기대했던 것과 달리, Sigmoid + MSE 모델이 다중분류에 주로 쓰이는 Softmax + Cross-Entropy 함수보다 더 정확도가 높게 나왔다.

이유는 쉽게 찾을 수 있었는데, 실험한 모델이 단층 퍼셉트론이기 때문이다. 흔히 Sigmoid + MSE 보다 Softmax + Cross Entropy를 선호하는 이유는 Cost Function인 MSE와 Cross Entropy를 미분해보면 알 수 있다.

$$\frac{\partial}{\partial b_{ij}} J(W, b; x, y) = \frac{1}{n} \sum_{x} (\sigma(x) - y)\sigma'(z)$$

그림 6. MSE 미분 결과

$$\frac{\partial}{\partial b_{ij}} J(W, b; x, y) = \frac{1}{n} \sum_{x} (\sigma(z) - y)$$

그림 7. Cross Entropy 미분 결과

[1] https://curt-park.github.io/2018-09-19/loss-cross-entropy/

위 식에서 보이는 바와 같이 MSE 미분 결과에서 $\sigma'(z)$ 값이 있는 것을 확인할 수 있다. Sigmoid Function은 양 극점에서 미분 값이 0에 가까워지므로 loss를 잃게 되어 학습이 어려워진다는 단점이 있다. 하지만 Cross Entropy의 경우 그러한 과정이 없어 학습에 용이하다는 장점이 있다.

이러한 장단점으로 미루어 보았을 때, 이번 과제에 나온 모델이 단층 퍼셉트론이기 때문에 학습과정에서 Sigmoid의 단점이 적게 나타난 것으로 보인다. 하지만 다층 퍼셉트론으로 모델이 변경된다면 같은 Cost Function과 Activate Function에서 완전히 다른 결과가 나올 것으로 기대한다.

3.4 Cross-Entropy 손실함수 분석

우리는 Cross-Entropy를 베르누이 분포 함수에서 구할 수 있다.[1] 우리는 베르누이 확률변수의 분포를 베르누이 확률분포라고 부른다. 베르누이분포의 확률 질량 함수식은 다음과 같다. (여기서 μ는 1이 나올 확률을 의미한다.)

$$\mathrm{Bern}(x;\mu) = \begin{cases} \mu & (\text{if } x=1), \\ 1-\mu & (\text{if } x=0) \end{cases} \qquad \mathrm{Bern}(x;\mu)=\mu^{x}(1-\mu)^{(1-x)}$$

그림 8. 베르누이분포 확률 질량 함수

베르누이분포에 log를 취해주면 우리가 아는 Cross-Entropy 함수가 나온다. 이것이 의미하는 바는 Cross-Entropy 함수에서 Likelihood를 가장 크게 하는 μ를 추정할 수 있다는 것이다.

4 수정한 코드

Sigmoid + MSE에 해당하는 비교대상을 코드로 제작해 보았다.
수정한 부분은 노란색으로 표시하였으며, 2장 과제의 pulsar.ipynb 파일을 참고하여 제작하였다.

```
# -*- coding: utf-8 -*-
# %run ../chap01/abalone.ipynb

def steel_exec(epoch_count=10, mb_size=10, report=1, learning_rate = 0.001):
    load_steel_dataset()
    init_model()
    train_and_test(epoch_count, mb_size, report, learning_rate)

def load_steel_dataset():
    with open('../../data/chap03/faults.csv') as csvfile:
        csvreader = csv.reader(csvfile)
        next(csvreader, None)
```

[1] https://curt-park.github.io/2018-09-19/loss-cross-entropy/

3장. 철판 분류 신경망 과제

```
        rows = []
        for row in csvreader:
            rows.append(row)

    global data, input_cnt, output_cnt
    input_cnt, output_cnt = 27, 7
    data = np.asarray(rows, dtype='float32')

def forward_postproc(output, y):
    entropy = sigmoid_mean_squared_error_with_logits(y, output)
    loss = np.mean(entropy)
    return loss, [y, output, entropy]

def backprop_postproc(G_loss, aux):
    y, output, entropy = aux

    g_loss_entropy = 1.0 / np.prod(entropy.shape)
    g_entropy_output = sigmoid_mean_squared_error_with_logits_derv(y, output)


    G_entropy = g_loss_entropy * G_loss
    G_output = g_entropy_output * G_entropy

    return G_output

def eval_accuracy(output, y):
    estimate = np.greater(output, 0)
    answer = np.greater(y, 0.5)
    correct = np.equal(estimate, answer)

    return np.mean(correct)

def relu(x):
    return np.maximum(x, 0)

def sigmoid(x):
    return np.exp(-relu(-x)) / (1.0 + np.exp(-np.abs(x)))

def sigmoid_derv(x, y):
    return y * (1 - y)

def sigmoid_mean_squared_error_with_logits(labels, logits):
    probs = sigmoid(logits)
    return np.sum(np.square(labels - probs), axis=1)

def sigmoid_mean_squared_error_with_logits_derv(z, x):
    return -z + sigmoid(x)
```

[1] https://curt-park.github.io/2018-09-19/loss-cross-entropy/