2021-2 딥러닝응용기초 : 프로젝트 #1/4

IT공학과
2031695
배현진

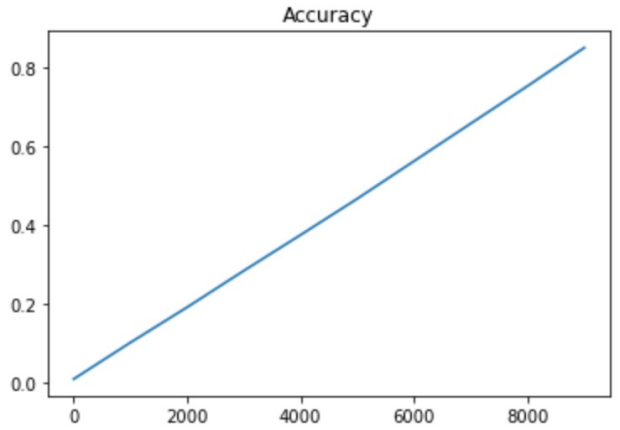
실험 코드

```
def predict(network, x):
    w1, w2, w3 = network['W1'], network['W2'], network['W3']
    b1, b2, b3 = network['b1'], network['b2'], network['b3']

    a1 = np.dot(x, w1) + b1
    z1 = 활성화함수(a1)
    a2 = np.dot(z1, w2) + b2
    z2 = 활성화함수(a2)
    a3 = np.dot(z2, w3) + b3
    y = softmax(a3)

    return y

x, t = get_data()
network = init_network()

batch_size = 100 # 배치 크기
accuracy_cnt = 0

step = list()
accuracy = list()
for i in range(0, len(x), batch_size):
    x_batch = x[i:i+batch_size]
    y_batch = predict(network, x_batch)
    p = np.argmax(y_batch, axis=1)
    accuracy_cnt += np.sum(p == t[i:i+batch_size])

    if i % 1000 == 0:
        step.append(i)
        accuracy.append(float(accuracy_cnt) / len(x))
```

- 다음과 같은 코드를 이용해 mnist 숫자 손글씨 인식 시 활성화 함수가 예측 성능에 영향을 미치는지를 실험

1) 항등 함수(Identity Function)

- 입력을 그대로 출력해 입력과 출력이 같다.

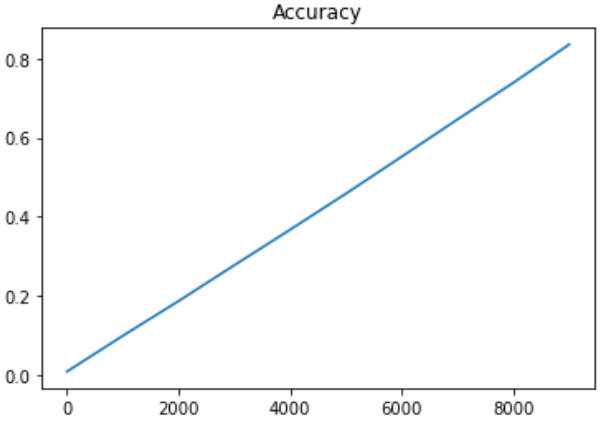



```
step 1000 : 0.009600
step 2000 : 0.102400
step 3000 : 0.192100
step 4000 : 0.285000
step 5000 : 0.375900
step 6000 : 0.468000
step 7000 : 0.563600
step 8000 : 0.659100
step 9000 : 0.754300
step 10000 : 0.851600
```

2) 계단 함수(Step Function)

- 수식: $$h(x) = \begin{cases} 0, x \le 0 \\ 1, x > 0 \end{cases}$$

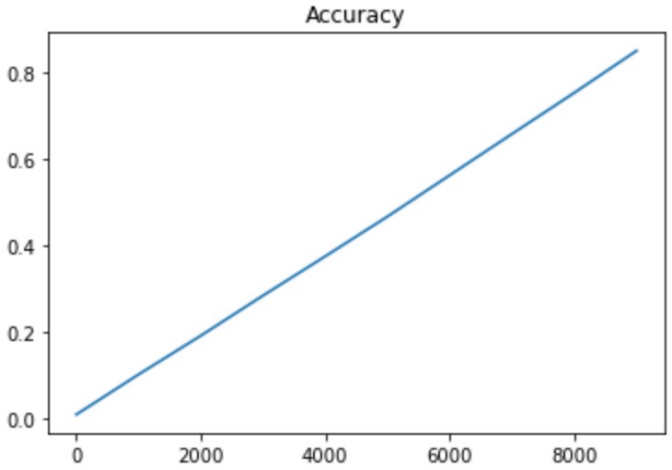
- 0을 기준으로 출력을 결정짓기 때문에 다중 출력이 불가능하다.




```
step 1000 : 0.009500
step 2000 : 0.099800
step 3000 : 0.187500
step 4000 : 0.278100
step 5000 : 0.367800
step 6000 : 0.458800
step 7000 : 0.552200
step 8000 : 0.646200
step 9000 : 0.738700
step 10000 : 0.835300
```

3) 시그모이드 함수(Sigmoid Function)

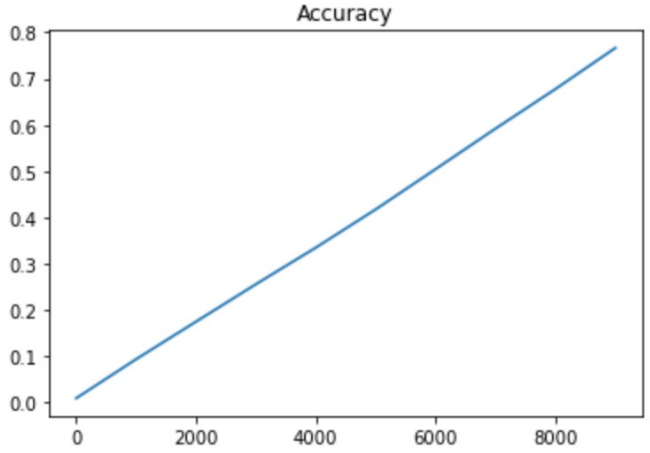
- 수식: $\mathrm{h}(\mathrm{x}) = \frac{1}{1+\exp(-x)}$
- 입력에 따라 값이 급격하게 변하지 않고, 출력값의 범위가 0과 1 사이로 제한됨으로써 exploding gradient를 방지할 수 있지만, gradient 값이 0에 수렴하는 gradient vanishing이나 gradient 업데이트 중 지그재그로 변동하는 문제점이 발생한다.




```
step 1000 : 0.009600
step 2000 : 0.102400
step 3000 : 0.192100
step 4000 : 0.285000
step 5000 : 0.375900
step 6000 : 0.468000
step 7000 : 0.563600
step 8000 : 0.659100
step 9000 : 0.754300
step 10000 : 0.851600
```

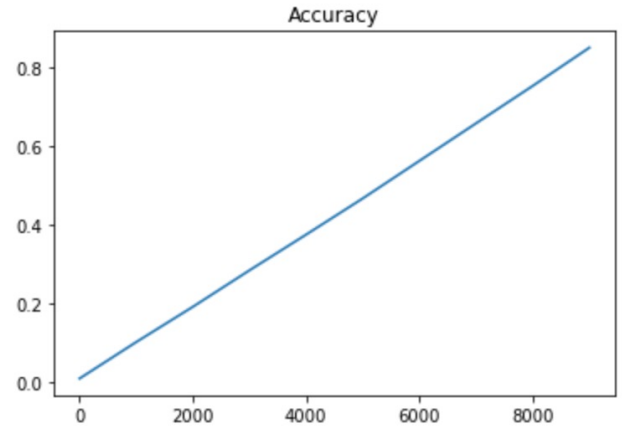
4) ReLu 함수

- 수식: $h(x) = \begin{cases} 0, x \leq 0 \\ 1, x > 0 \end{cases}$
- 값이 0보다 작거나 같으면 0, 0보다 크면 선형 함수에 값을 대입해 다른 활성화 함수에 비해 효율적인 결과를 보이지만 음수는 학습하지 못하는 단점이 있다.

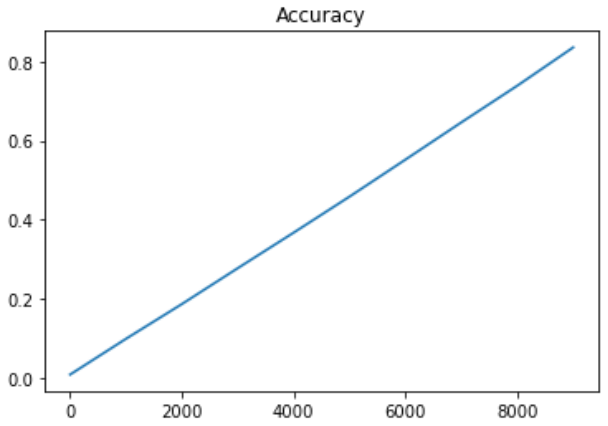



```
step 1000 : 0.008900
step 2000 : 0.093300
step 3000 : 0.174500
step 4000 : 0.255500
step 5000 : 0.334600
step 6000 : 0.417300
step 7000 : 0.504400
step 8000 : 0.592000
step 9000 : 0.677500
step 10000 : 0.766400
```

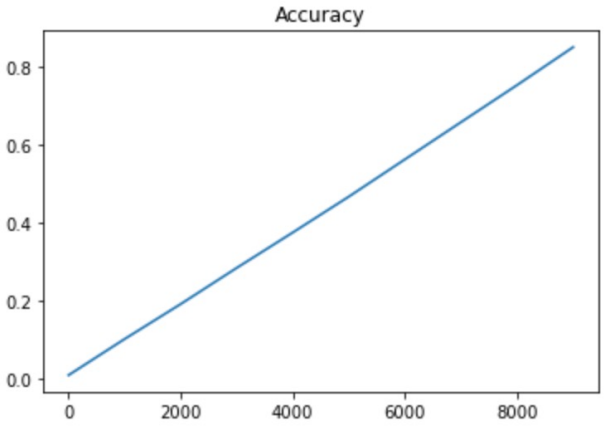
결과



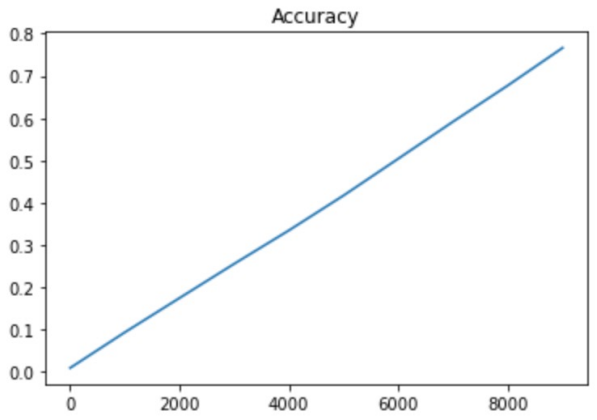
항등함수



계단함수



시그모이드



ReLU

- 기계학습 시 activation function 별 큰 차이는 없다. 하지만 딥러닝을 사용해 모델이 깊어지게 된다면 항등함수나 계단함수의 경우 미분이 불가하기 때문에 비선형 함수를 사용해야 할 것이며, 이에 따른 활성화 함수 별 성능 차이가 발생할 것이다.