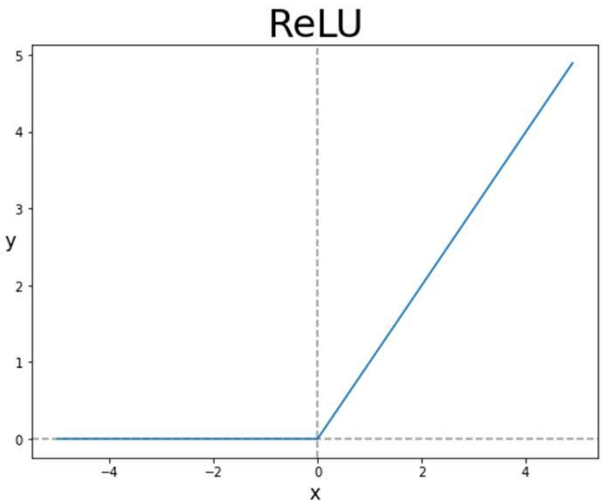
Week 6. Backpropagation(5.5~5.6)

21기 분석 이건하

활성화 함수 계층 구현하기(ReLu)

- ReLu
  - ReLu 수식
    - 입력값이 0 이상이면 입력값을 그대로 출력
    - 입력값이 0 이하면 0을 출력

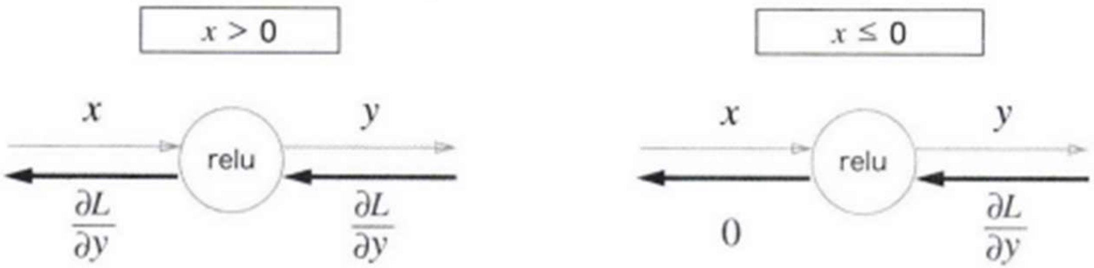


$$y = \begin{cases} x & (x > 0) \\ 0 & (x \le 0) \end{cases}$$

$$\frac{\partial y}{\partial x} = \begin{cases} 1 & (x > 0) \\ 0 & (x \le 0) \end{cases}$$

  - ReLu 계층의 계산 그래프
    - 순전파 때의 입력인 x가 0 초과면 역전파는 상류의 값을 그대로 하류로
    - 순전파 때의 x가 0 이하면 역전파 때는 하류로 신호를 보내지 않음 (0을 전달)

그림 5-18 ReLU 계층의 계산 그래프



활성화 함수 계층 구현하기(ReLu)

- ReLu 계층의 계산 그래프
  - mask는 True/False로 구성된 numpy 배열

```python
class Relu:
    def __init__(self):
        self.mask = None

    def forward(self,x):
        self.mask = (x<=0)
        out = x.copy()
        out[self.mask] = 0
        return out

    def backward(self,dout):
        dout[self.mask] = 0
        dx = dout
        return dx
```

(x가 0이하면 Ture, 크면 False로 변환)

```python
x = np.array([[1.0,-0.5], [-2.0,3.0]])
print(x)

mask = (x<=0)
print(mask)
```

✓ 0.0s

```
[[ 1.  -0.5]
 [-2.   3. ]]
[[False  True]
 [ True False]]
```

```python
x[mask]
```

✓ 0.0s

```
array([-0.5, -2. ])
```

(음수인 값의 인덱스만 선택)

```python
x[mask] = 0
x
```

✓ 0.0s

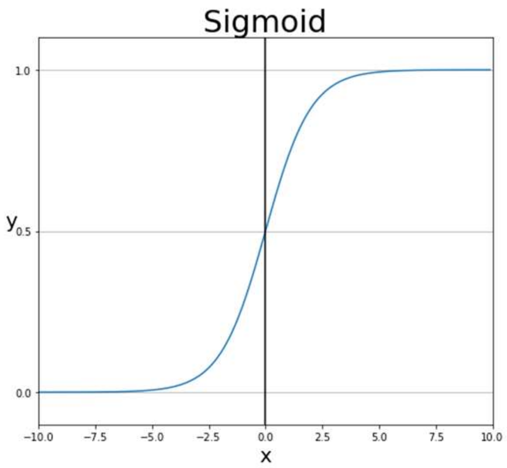
```
array([[1., 0.],
       [0., 3.]])
```

(0으로 변환)

BOAZ

활성화 함수 계층 구현하기(Sigmoid)

- Sigmoid
  - Sigmoid 수식
    - 입력값이 클수록 1에 가까운 값 출력
    - 입력값이 작을수록 0에 가까운 값 출력



$$y = \frac{1}{1 + \exp(-x)}$$

  - Sigmoid 계층의 계산 그래프
    - '/': 역전파 때는 상류에서 흘러온 값에 $-y^2$를 곱해서 하류로 전달
    - 'exp': 역전파 때의 미분값은 exp(x)

그림 5-21 Sigmoid 계층의 계산 그래프(간소화 버전)

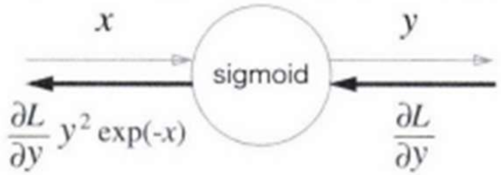
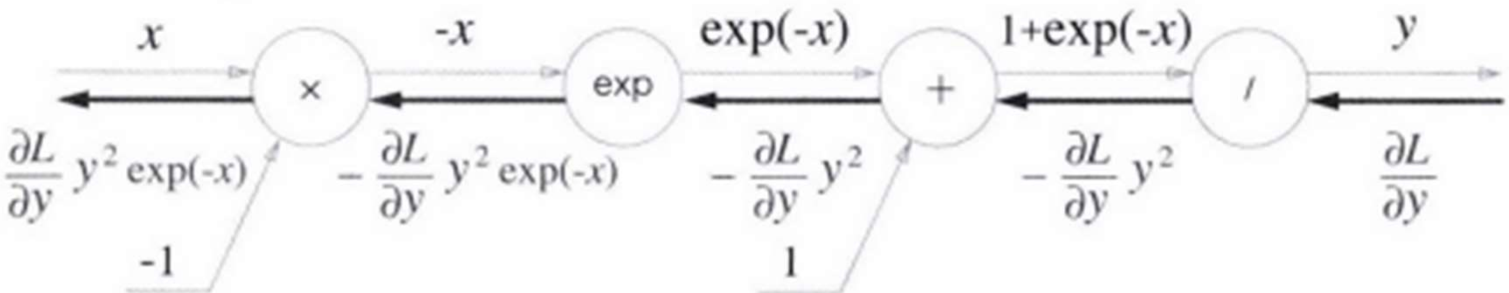


그림 5-20 Sigmoid 계층의 계산 그래프



활성화 함수 계층 구현하기(Sigmoid)

- Sigmoid 계층의 계산 그래프
  - 코드 구현

```python
class Sigmoid:
    def __init__(self):
        self.out = None

    def forward(self,x):
        out = 1 / (1+np.exp(-x))
        self.out = out
        return out

    def backward(self,dout):
        dx = dout*(1.0-self.out)*self.out
        return dx
```

(역전파 계층 정리)

$$\frac{\partial L}{\partial y} y^2 \exp(-x) = \frac{\partial L}{\partial y} \frac{1}{(1+\exp(-x))^2} \exp(-x)$$

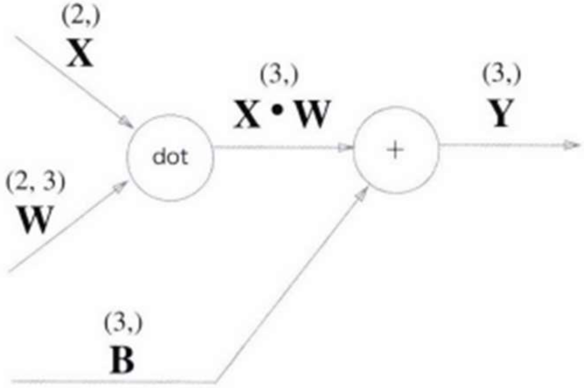
$$= \frac{\partial L}{\partial y} \frac{1}{1+\exp(-x)} \frac{\exp(-x)}{1+\exp(-x)}$$

$$= \frac{\partial L}{\partial y} y(1-y)$$

Affine 계층의 계산 그래프

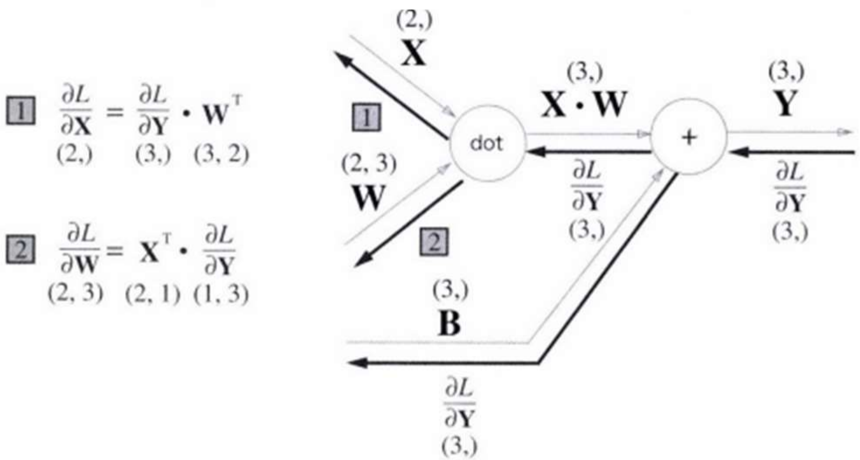
- Affine 계층 **(흐르는 값이 '스칼라(단일값)'에서 '행렬'로 바뀐 것)**
  - 신경망의 순전파에서 뉴런의 가중치 합을 구할 때 Y = np.dot(X,W)+B 로 계산했었다.
    - 신경망의 순전파 때 수행하는 행렬의 곱을 Affine transformation 이라고 한다.
    - Affine transformation 을 수행하는 처리를 'Affine 계층'이라는 이름으로 구현

그림 5-24 Affine 계층의 계산 그래프



  - Affine 계층의 역전파
    - X와 $\underset{(2,)}{\frac{\partial L}{\partial \mathbf{X}}}$ 의 형상이 같고, W와 $\underset{(2,\ 3)}{\frac{\partial L}{\partial \mathbf{W}}}$ 의 형상이 같음
    - 행렬 곱(dot노드)의 역전파 단계에서는, 순전파의 입력방향과 반대로 곱을 진행하기 때문에, 대응하는 차원의 원소 수가 일치하도록 전치행렬(.T)로 변환

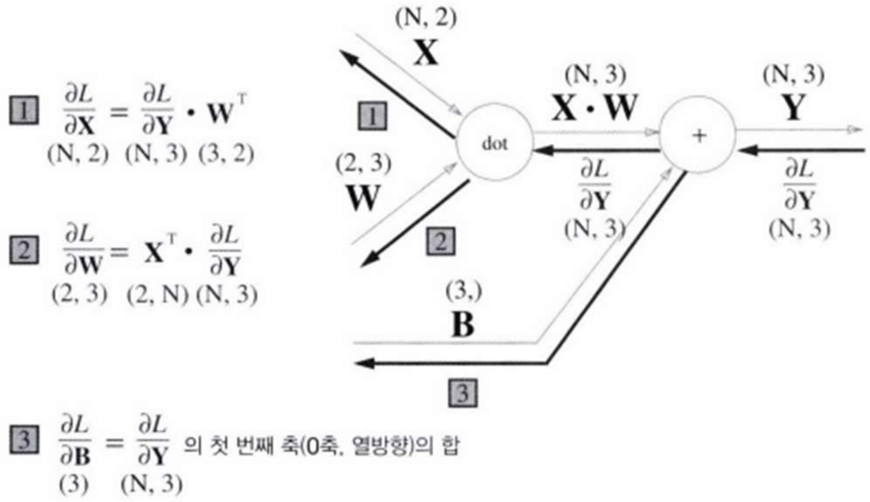
그림 5-25 Affine 계층의 역전파



배치용 Affine 계층의 계산 그래프

- 배치용 Affine 계층
  - N개의 데이터를 묶어서 순전파 하는 경우(배치용 Affine 계층)
    - X의 형상이 (1,2)에서 (N,2)으로 변경 (N은 배치 크기)

그림 5-27 배치용 Affine 계층의 계산 그래프



  - 배치용 Affine 계층의 편향의 덧셈
    - **순전파** 때의 편향의 덧셈은 각각의 데이터(열)에 더해짐 (1개->N개)
    - **역전파** 때의 편향은, 각 데이터에 대한 미분값이 편향의 원소에 모임(SUM)

```
X_dot_w = np.array([[0,0,0], [10,10,10]])
B = np.array([1,2,3])

X_dot_w+B
✓ 0.0s
array([[ 1,  2,  3],
       [11, 12, 13]])
```

순전파

```
dY = np.array([[1,2,3],[4,5,6]])
dB = np.sum(dY,axis=0)
dB
✓ 0.0s
array([5, 7, 9])
```

역전파

Affine 계층의 계산 그래프

- Affine 계층의 계산그래프
  - 코드 구현

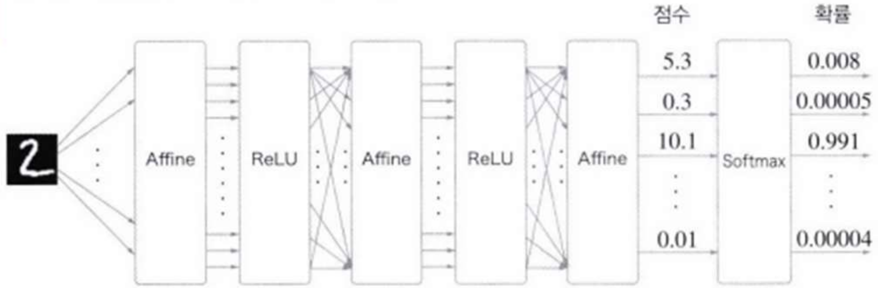
```python
class Affine:
    def __init__(self,W,b):
        self.W = W
        self.b = b
        self.x = None
        self.dW = None
        self.db = None

    def forward(self,x):
        self.x = x
        out = np.dot(x,self.W)+self.b
        return out

    def backward(self,dout):
        dx = np.dot(dout,self.W.T)
        self.dW = np.dot(self.x.T, dout)
        self.db = np.sum(dout,axis=0)
        return dx
```

Softmax-with-Loss 계층 구현하기

그림 5-28 입력 이미지가 Affine 계층과 ReLU 계층을 통과하며 변환되고, 마지막 Softmax 계층에 의해서 10개의 입력이 정규화된다. 이 그림에서는 숫자 '0'의 점수는 5.3이며, 이것이 Softmax 계층에 의해서 0.008(0.8%)로 변환된다. 또, '2'의 점수는 10.1에서 0.991(99.1%)로 변환된다.

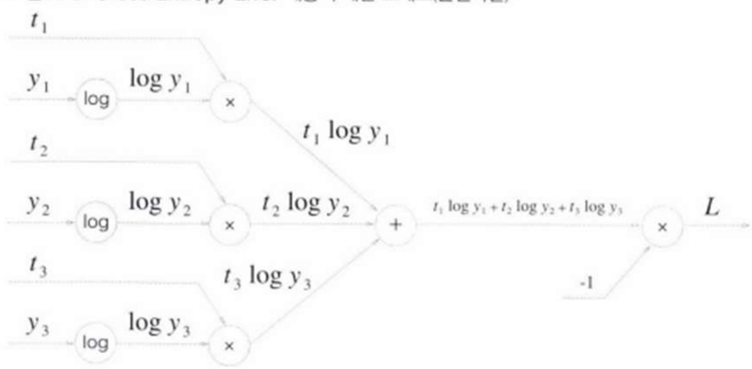


- 용어복습
  - Softmax
    - 소프트맥스 함수는 입력값을 정규화하여 출력
    - Affine 계층의 마지막 출력(Score)을 클래스에 대한 확률값으로 변환하는 활성화 함수

  - Cross Entropy Error
    - softmax함수를 통과한 최종 출력값(예측값)과 정답 레이블의 오차(Loss)를 계산
    - 시그마로 표현된 각 오차의 합이지만, 실질적으로는 원핫인코딩 형태에 따른 정답레이블에 대한 오차만 계산 (정답레이블이 아닌 값은 0이 곱해지기 때문)

$$L = -\sum_k t_k \log y_k$$

그림 A-3 Cross Entropy Error 계층의 계산 그래프(순전파만)



BOAZ

Softmax-with-Loss 계층 구현하기

- Softmax-with-Loss 계층(Softmax 계층+Cross-Entropy-Error계층)
  - 순전파
    - 소프트맥스: 최종 출력값에 밑이 e인 자연로그 exp()를 곱하고, 그 값들의 합으로 나누는 과정 (y 출력)
    - 교차 엔트로피 오차: 소프트맥스를 통과한 y값에 -log를 곱하고, 정답레이블의 동일한 인덱스값 t를 곱한 값의 합 (실질적으로는 정답레이블에 대한 오차만 산정)

그림 A-2 Softmax 계층의 계산 그래프(순전파만)

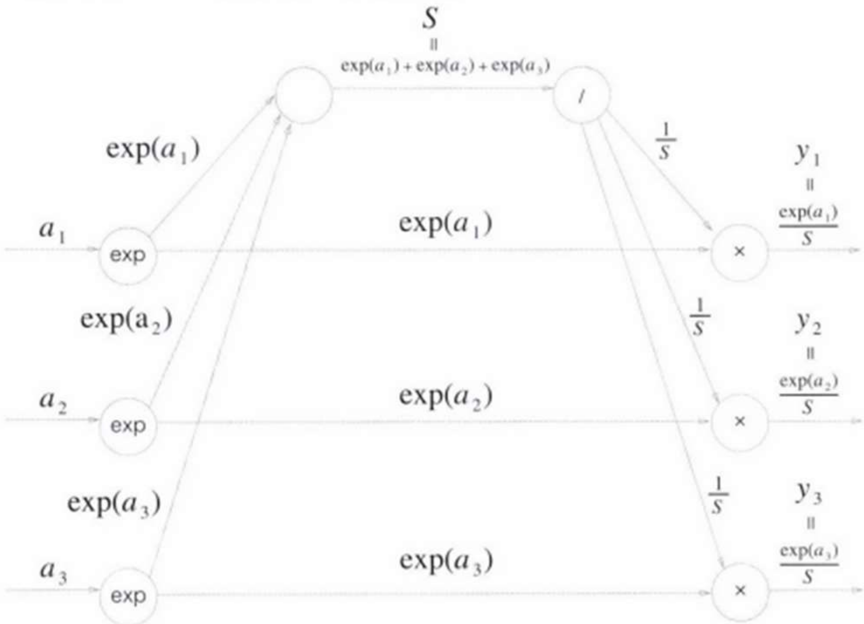
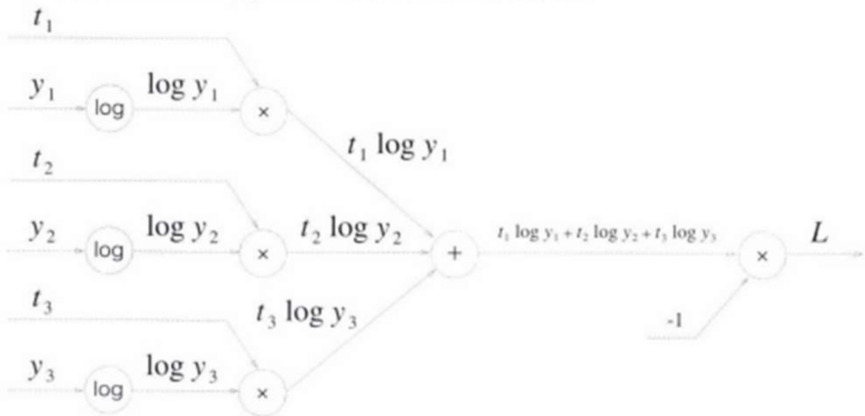


그림 A-3 Cross Entropy Error 계층의 계산 그래프(순전파만)



Softmax-with-Loss 계층 구현하기

- Softmax-with-Loss 계층 (Softmax 계층+Cross-Entropy-Error계층)
  - 역전파 (결과 도출 과정은 p294~298 참조)
    - softmax 계층은 (y1,y2,y3)를 출력, 정답레이블은 (t1,t2,t3) 형태
    - 역전파의 결과는 (y1-y1, y2-y2, y3-y3) 형태 (각각에 대한 차분이 쉽게 산정)

그림 A-1 Softmax-with-Loss 계층의 계산 그래프

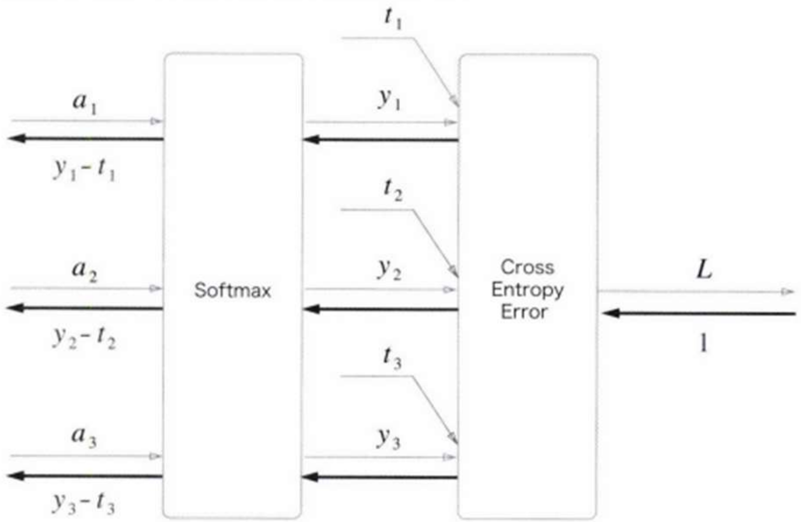
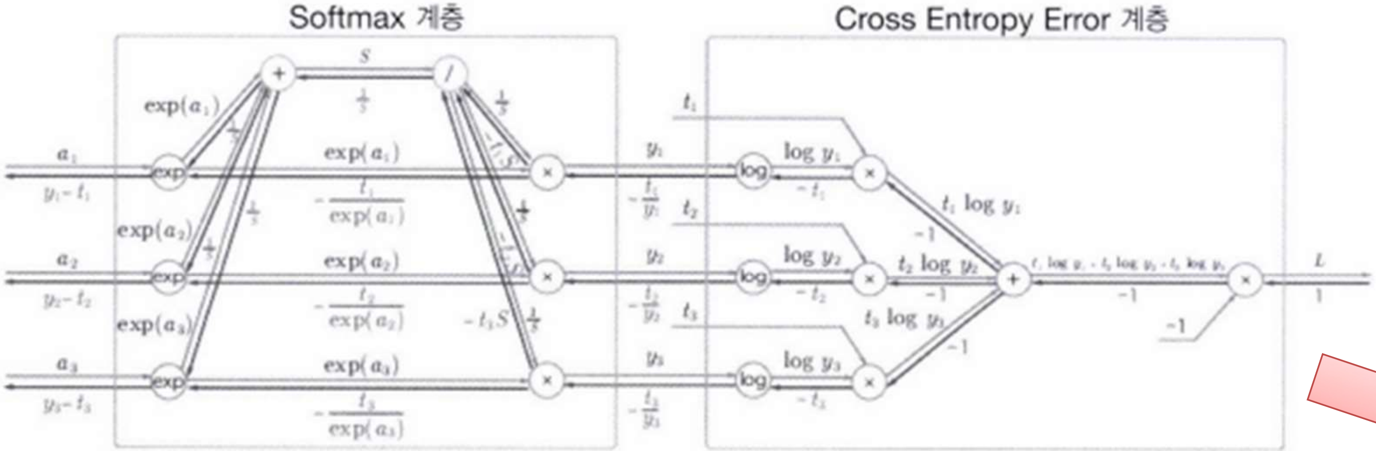


그림 5-29 Softmax-with-Loss 계층의 계산 그래프



$$\left(\frac{1}{S}-\frac{t_1}{\exp(a_1)}\right)\exp(a_1)$$

Softmax-with-Loss 계층 구현하기

- Softmax-with-Loss 계층의 계산그래프
  - 코드구현

```
class SofrmaxWithLoss:
    def __init__(self):
        self.loss = None #손실
        self.y = None #softmax의 출력
        self.t = None #정답 레이블(원-핫 벡터)

    def forward(self,x,t):
        self.t = t
        self.y = softmax(x)
        self.loss = cross_entropy_error(self.y,self.t)
        return self.loss

    def backward(self,dout=1):
        batch_size = self.t.shape[0]
        dx = (self.y - self.t) / batch_size
        return dx
```

예: 정답레이블이 (0,1,0)인 데이터에 대해, 소프트맥스 계층이 (0.3, 0.2, 0.5)를 출력했을때,

- Softmax-with-Loss 계층의 순전파의 결과(Loss)는 '-1.6094379124341003' (정답레이블만으로 산정)

- Softmax-with-Loss 계층의 역전파는 (y1-t1, y2-t2, y3-t3이므로, (0.3, -0.8, 0.5)을 전달

(순전파 과정의 최종결과인 Loss는 정답레이블에 대해서만 계산했지만, 역전파에서는 정답이 아닌 레이블에 대한 값도 사용된다)