큰 learning rate 사용하는게 좋음.

learning rate가 커지면 → 파라미터의 범위가 커짐 ⟶ gradient가 증폭 ⇒ model ↑
gradient exploding

이유

[BN case]

$$\frac{\partial BN((aW)u)}{\partial u} = \frac{\partial BN(Wu)}{\partial u} \qquad \frac{\partial BN((aW)u)}{\partial (aW)} = \frac{1}{a} \cdot \frac{\partial BN(Wu)}{\partial W}$$

⇒ 평균과 분산에 의한 normalization으로 미분값이 작아짐. learning rate를 올려도 gradient가 급격하게 증가하지 않음

④ 지수가중이동평균

- 과거의 것을 잊어버림
- 지수적으로 감소하는 가중치를 곱해줌

$$V_t = \beta V_{t-1} + (1-\beta)\theta_t$$

$$V_0 = 0$$
$$V_1 = 0.9V_0 + 0.1\theta_1$$
$$V_2 = 0.9V_1 + 0.1\theta_2$$
$$\vdots$$
$$V_t = 0.9V_{t-1} + 0.1\theta_t$$

세타들의 정보 작게봄

$$V_t = \beta V_{t-1} + (1-\beta)\theta_t$$

$\beta = 0.9$ V_t는 평균에 해당

$$\approx \frac{1}{1-\beta} \text{ days}$$

$\Rightarrow \beta = 0.9 \Rightarrow \frac{1}{1-0.9} = 10$

$\beta = 0.98 \Rightarrow \frac{1}{1-0.98} = 50$

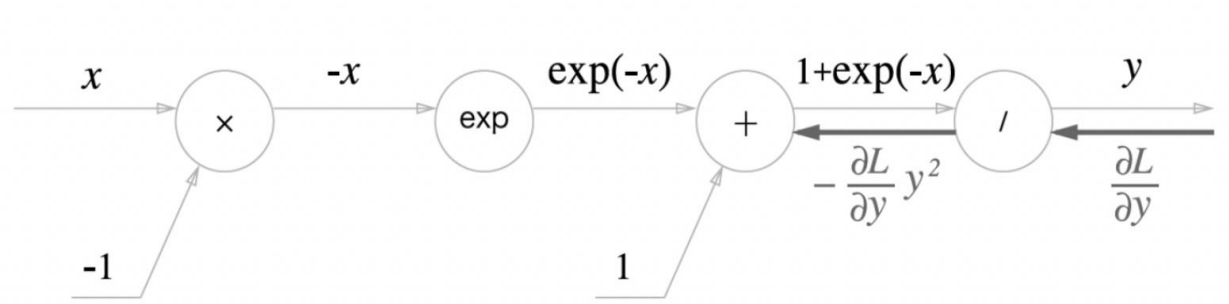
β값이 클수록 더 많은 기간의 평균을 이용하므로 부드러워짐

1. 소프트맥스가 (0.3, 0.2, 0.5)를 출력했다고 할 때, 소프트맥스 계층의 역전파는 (0.3, -0.8, 0.5)로 앞 계층에 큰 오차를 전파하게 됨. 결과적으로 Softmax 계층의 앞 계층들은

2. 소프트맥스가 (0.01, 0.99, 0.)을 출력했다면 역전파는 (0.01, -0.01, 0) 으로 오차가 작아짐. 이번에는 앞 계층으로 전달된 오차가 작으므로 학습하는 정도도 작아짐!

.3]:
```python
class SoftmaxWithLoss:
    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)  # 3.5.2, 4.2.2에서 구현
        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
```

5.7 오차역전파법 구현하기



2단계

- 노드는 상류의 값을 여과 없이 하류로 보내므로 다음과 같다.