Backward Propagation of Errors

Hee-il Hahn

Professor
Department of Information and Communications Engineering
Hankuk University of Foreign Studies
hihahn@hufs.ac.kr

Chain Rules

- Chain Rules and Computation Graph

$$z = t^2$$

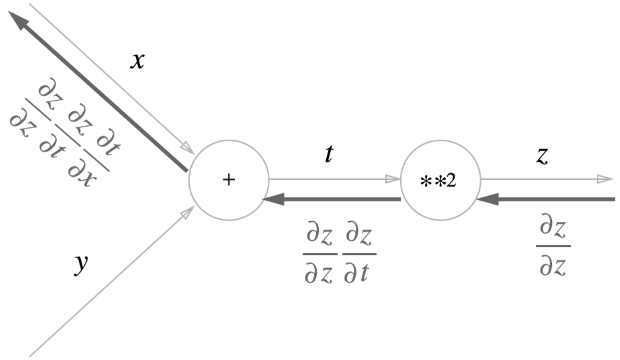
$$t = x + y$$

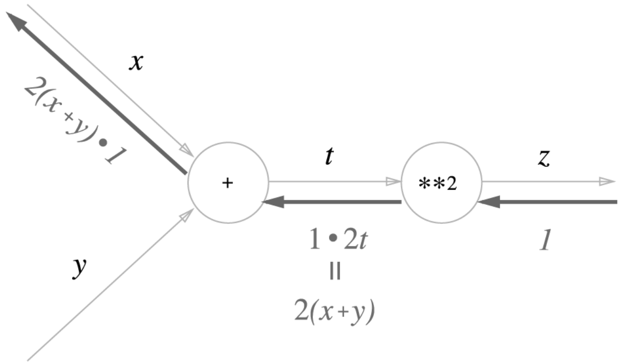
$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial t}\frac{\partial t}{\partial x}$$

$$\frac{\partial z}{\partial t} = 2t$$

$$\frac{\partial t}{\partial x} = 1$$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial t}\frac{\partial t}{\partial x} = 2t \cdot 1 = 2(x+y)$$





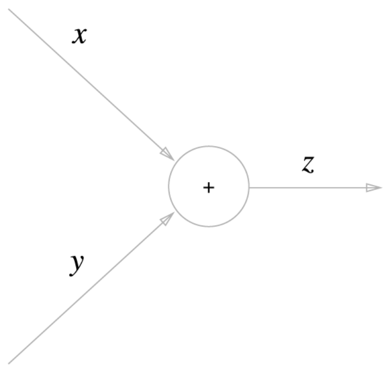
Backwrd Propagation

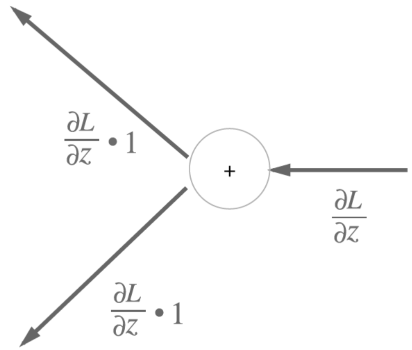
- Additive Nodes

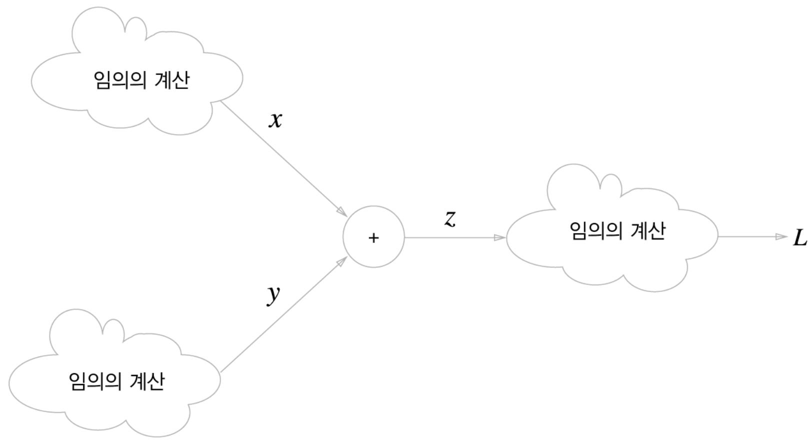
$$z = x + y$$

$$\frac{\partial z}{\partial x} = 1$$

$$\frac{\partial z}{\partial y} = 1$$







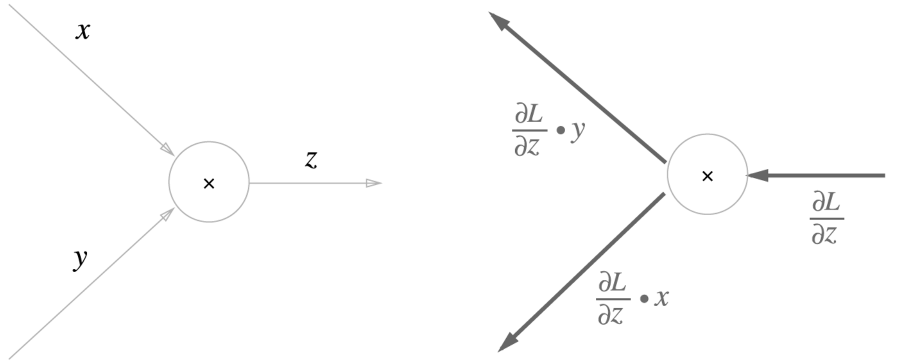
Backwrd Propagation – cont.

- Multiplicative Nodes

$$z = xy$$

$$\frac{\partial z}{\partial x} = y$$

$$\frac{\partial z}{\partial y} = x$$



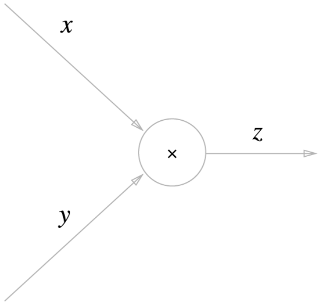
Implementation of Simple Layers

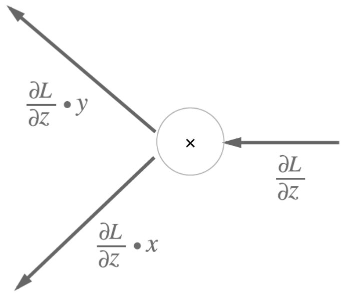
- Multiplication Layer

$$z = xy$$

$$\frac{\partial z}{\partial x} = y$$

$$\frac{\partial z}{\partial y} = x$$






```python
class MulLayer:
    def __init__(self):
        self.x = None
        self.y = None

    def forward(self, x, y):
        self.x = x
        self.y = y
        out = x * y

        return out

    def backward(self, dout):
        dx = dout * self.y  # x와 y를 바꾼다.
        dy = dout * self.x

        return dx, dy
```

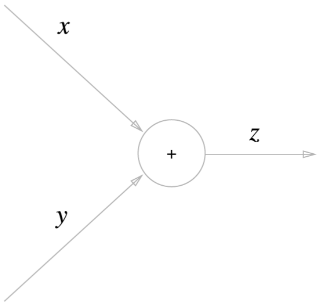
Implementation of Simple Layers – cont.

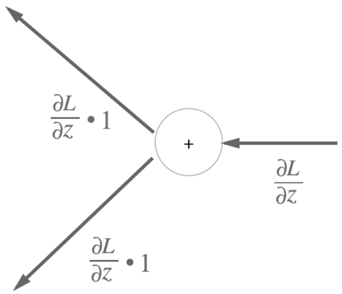
- Addition Layer

$$z = x + y$$

$$\frac{\partial z}{\partial x} = 1$$

$$\frac{\partial z}{\partial y} = 1$$





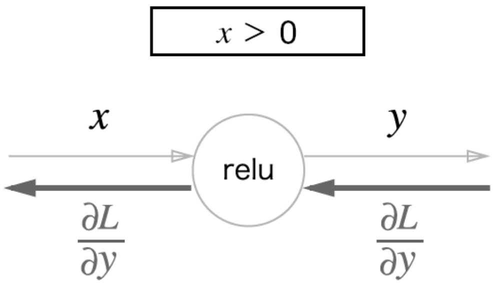

```
class AddLayer:
    def __init__(self):
        pass

    def forward(self, x, y):
        out = x + y

        return out

    def backward(self, dout):
        dx = dout * 1
        dy = dout * 1

        return dx, dy
```

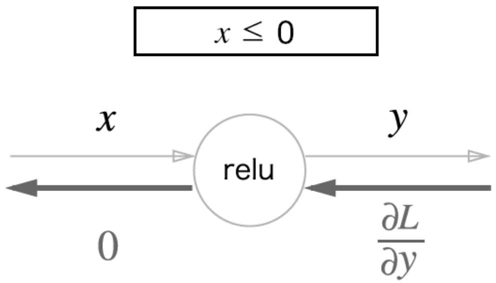
Implementation of Activation Function Layers

- Relu Layer

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

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



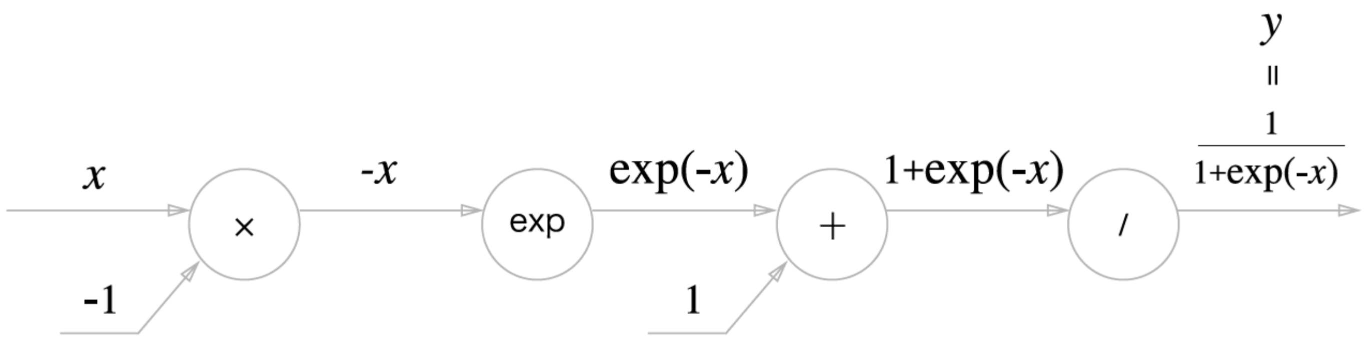



```python
class Relu:
    def __init__(self):
        self.mask = None

    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
```

Implementation of Activation Function Layers – cont.

- Sigmoid Layer (forward)

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

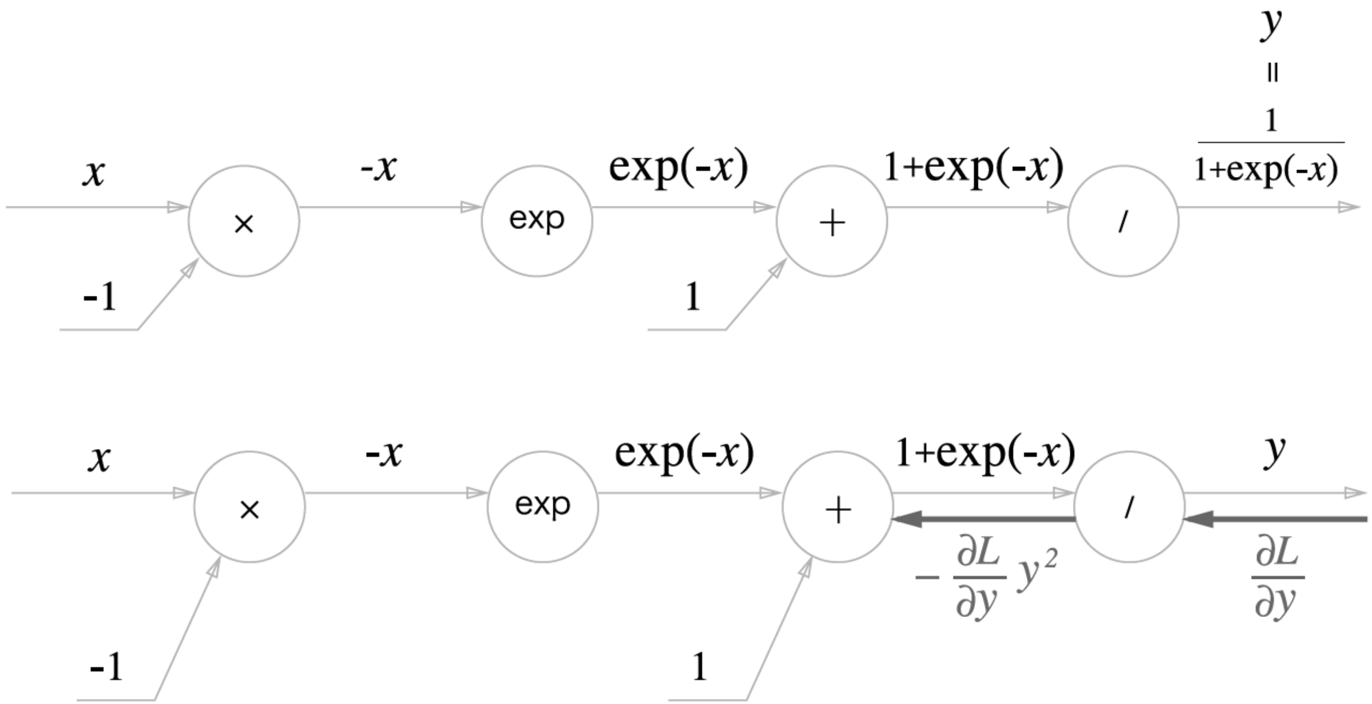


Implementation of Activation Function Layers – cont.

- Sigmoid Layer (backward)
  - 1 단계 ('/' 노드: $y = \frac{1}{x}$ 미분)

$$\frac{\partial y}{\partial x} = -\frac{1}{x^2}$$

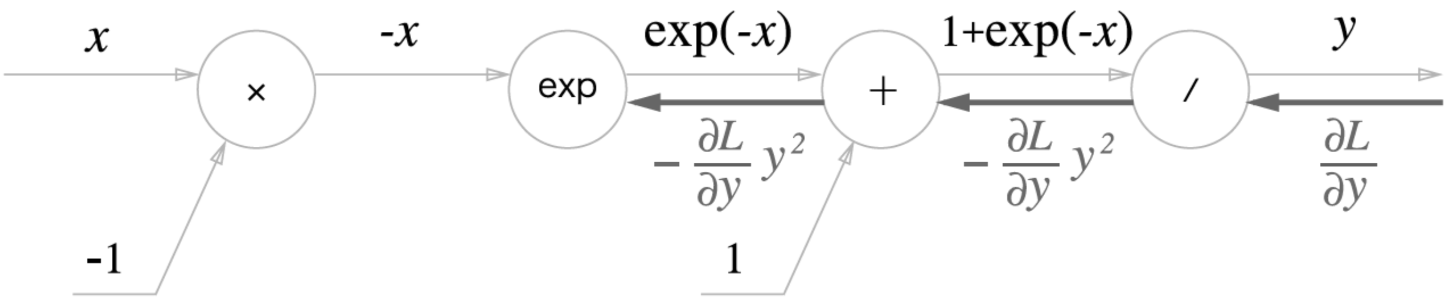
$$= -y^2$$



Implementation of Activation Function Layers – cont.

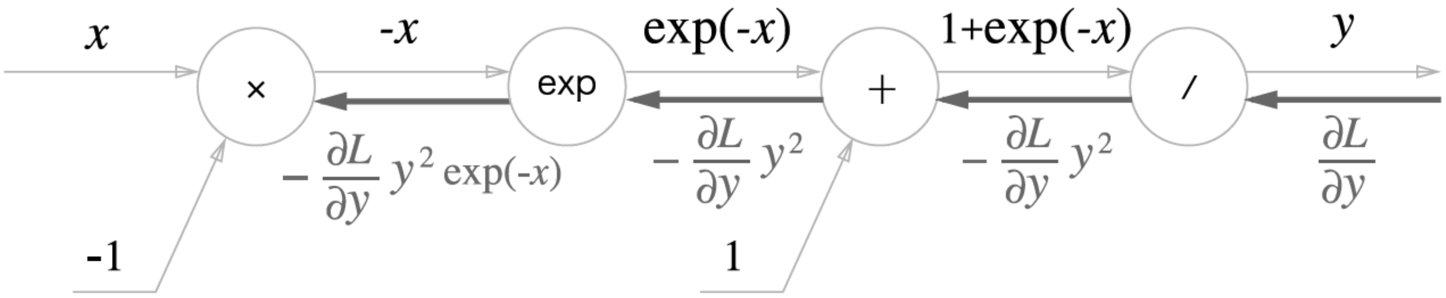
- Sigmoid Layer (backward)

  - 2 단계 ('+' 노드)



Implementation of Activation Function Layers – cont.

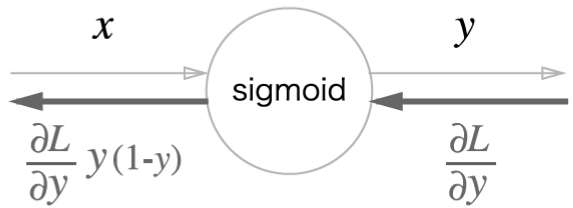
- Sigmoid Layer (backward)
  - 3 단계 ('exp' 노드)



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

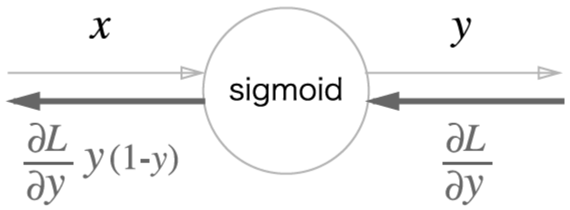
$$= \frac{\partial L}{\partial y} \frac{1}{1+\exp(-x)} \frac{\exp(-x)}{1+\exp(-x)}$$

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



Implementation of Activation Function Layers – cont.

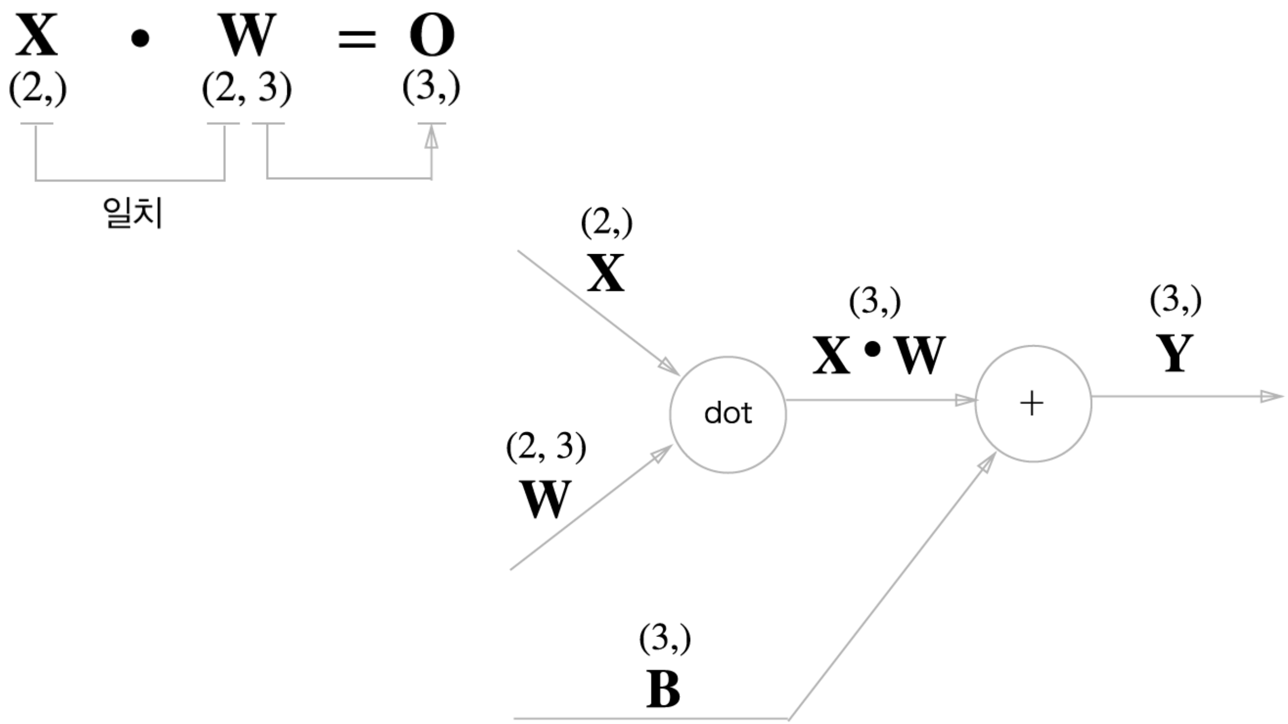
- Sigmoid Layer (backward)




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

    def forward(self, x):
        out = sigmoid(x)
        self.out = out
        return out

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

        return dx
```

Implementation of Affine/Softmax Layers – cont.

- Affine Layer (forward)



Implementation of Affine/Softmax Layers – cont.

- Affine Layer (backward, incremental process)

$$\frac{\partial L}{\partial \mathbf{X}} = \frac{\partial L}{\partial \mathbf{Y}} \cdot \mathbf{W}^{\mathrm{T}}$$

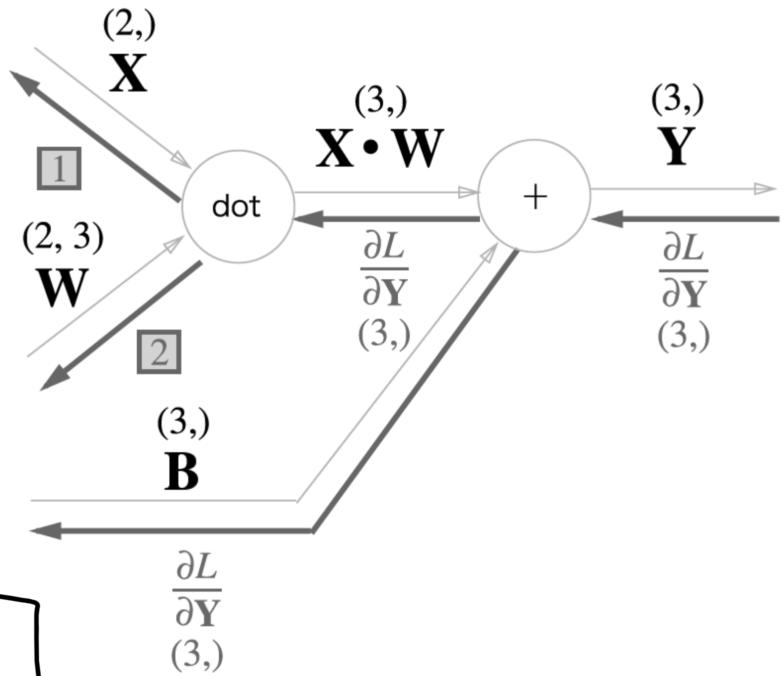
$$\frac{\partial L}{\partial \mathbf{W}} = \mathbf{X}^{\mathrm{T}} \cdot \frac{\partial L}{\partial \mathbf{Y}}$$

$$\mathrm{W} = \begin{pmatrix} w_{11} & w_{21} & w_{31} \\ w_{12} & w_{22} & w_{32} \end{pmatrix}$$

$$\mathrm{W}^{\mathrm{T}} = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{pmatrix}$$

$$\boxed{1} \quad \underset{(2,)}{\frac{\partial L}{\partial \mathrm{X}}} = \underset{(3,)}{\frac{\partial L}{\partial \mathbf{Y}}} \quad \underset{(3,\,2)}{\mathbf{W}^{\mathrm{T}}}$$

$$\boxed{2} \quad \underset{(2,\,3)}{\frac{\partial L}{\partial \mathbf{W}}} = \underset{(2,\,1)}{\mathbf{X}^{\mathrm{T}}} \quad \underset{(1,\,3)}{\frac{\partial L}{\partial \mathbf{Y}}}$$



X와 $\dfrac{\partial L}{\partial X}$의 $shape$

$$\mathbf{X} = (x_0, x_1, \cdots, x_n)$$

$$\frac{\partial L}{\partial \mathbf{X}} = \left(\frac{\partial L}{\partial x_0}, \frac{\partial L}{\partial x_1}, \cdots, \frac{\partial L}{\partial x_n} \right)$$

Implementation of Affine/Softmax Layers – cont.

- Affine Layer (backward, batch process)

$$\frac{\partial L}{\partial \mathbf{X}} = \frac{\partial L}{\partial \mathbf{Y}} \cdot \mathbf{W}^{\mathrm{T}}$$

$$\frac{\partial L}{\partial \mathbf{W}} = \mathbf{X}^{\mathrm{T}} \cdot \frac{\partial L}{\partial \mathbf{Y}}$$

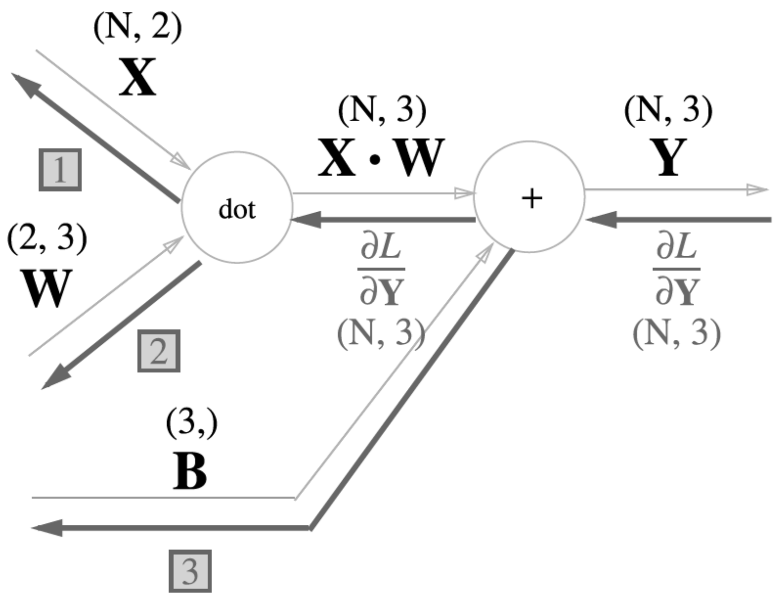
$$\mathrm{W} = \begin{pmatrix} w_{11} & w_{21} & w_{31} \\ w_{12} & w_{22} & w_{32} \end{pmatrix}$$

$$\mathrm{W}^{\mathrm{T}} = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{pmatrix}$$

$\boxed{1}$ $\underset{(N,\ 2)}{\frac{\partial L}{\partial \mathbf{X}}} = \underset{(N,\ 3)}{\frac{\partial L}{\partial \mathbf{Y}}} \bullet \underset{(3,\ 2)}{\mathbf{W}^{\mathrm{T}}}$

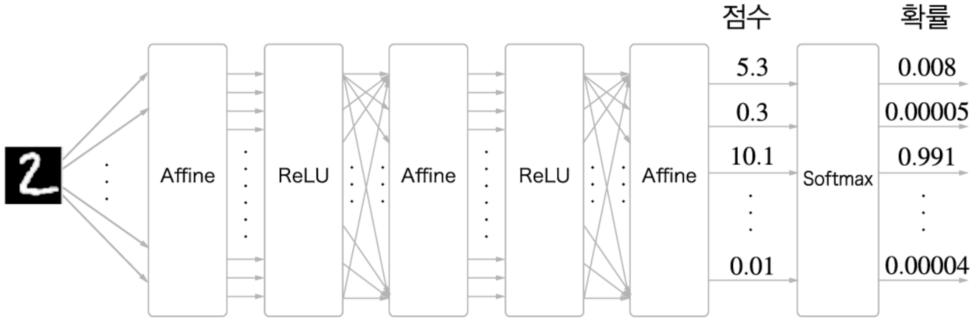
$\boxed{2}$ $\underset{(2,\ 3)}{\frac{\partial L}{\partial \mathbf{W}}} = \underset{(2,\ N)}{\mathbf{X}^{\mathrm{T}}} \bullet \underset{(N,\ 3)}{\frac{\partial L}{\partial \mathbf{Y}}}$

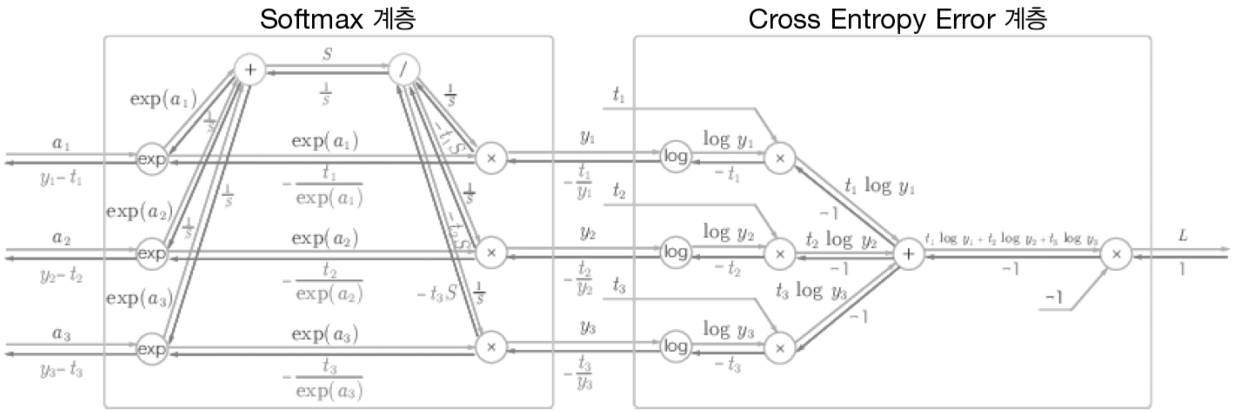
$\boxed{3}$ $\underset{(3)}{\frac{\partial L}{\partial \mathbf{B}}} = \underset{(N,\ 3)}{\frac{\partial L}{\partial \mathbf{Y}}}$ 의 첫 번째 축(0축, 열방향)의 합



Implementation of Affine/Softmax Layers – cont.

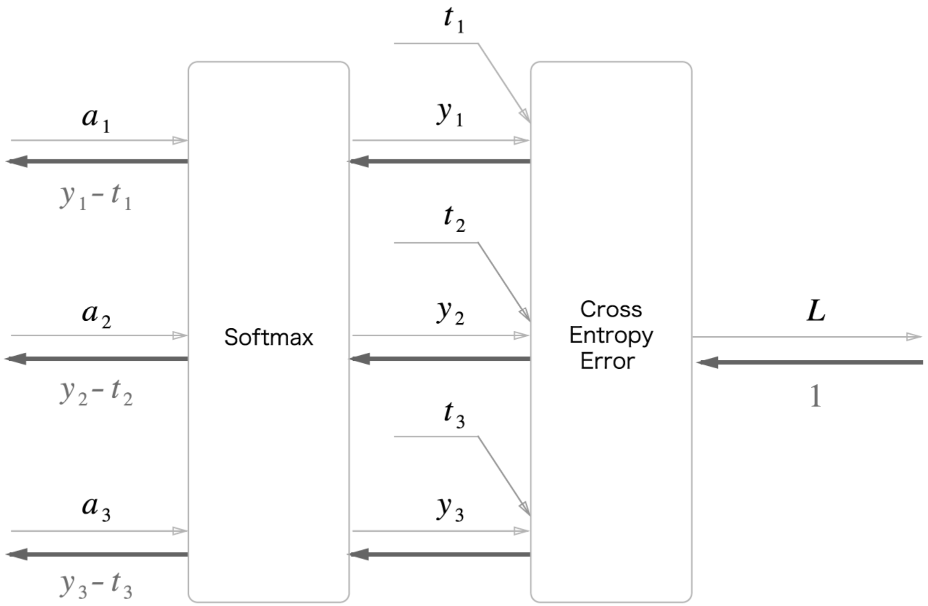
- Softmax-with-Loss Layer
  - MNIST 이미지 인식에 대한 softmax layer





Implementation of Affine/Softmax Layers – cont.

- Softmax-with-Loss Layer
  - MNIST 이미지 인식에 대한 softmax layer



$$L = -\sum_k t_k \log y_k \qquad y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n} \exp(a_i)}$$

$$\frac{\partial y_k}{\partial a_k} = y_k - y_k^2$$

$$\frac{\partial L}{\partial a_k} = \frac{\partial L}{\partial y_k}\frac{\partial y_k}{\partial a_k} + \sum_{j \neq k} \frac{\partial L}{\partial y_j}\frac{\partial y_j}{\partial a_k}$$

$$= -\frac{t_k}{y_k}\left(y_k - y_k^2\right) + \sum_{j \neq k}\left(-\frac{t_j}{y_j}\right)\left(-y_j y_k\right)$$

$$= -t_k + \sum_j t_j y_k = -t_k + y_k$$

Implementation of Affine/Softmax Layers – cont.

- Softmax-with-Loss Layer
  - MNIST 이미지 인식에 대한 softmax layer

```
class SoftmaxWithLoss:
    def __init__(self):
        self.loss = None # 손실함수
        self.y = None    # softmax의 출력
        self.t = None    # 정답 레이블(원-핫 인코딩 형태)

    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]
        if self.t.size == self.y.size: # 정답 레이블이 원-핫 인코딩 형태일 때
            dx = (self.y - self.t) / batch_size
        else:
            dx = self.y.copy()
            dx[np.arange(batch_size), self.t] -= 1
            dx = dx / batch_size

        return dx
```

```
def cross_entropy_error(y, t):
    if y.ndim ==1:
        t=t.reshape(1, t.size)
        y=y.reshape(1, y.size)

    batch_size = y.shape[0]
    delta = 1e-7
    return -np.sum(t*np.log(y+delta)) / batch_size
```

executed in 8ms, finished 15:32:41 2020-05-15

```
def identity_function(x):
    return x

def softmax(x):
    x = x - np.max(x) # 오버플로 대책
    return np.exp(x) / np.sum(np.exp(x))
```

Implementation of Two Layer NN

```python
class TwoLayerNet:

    def __init__(self, input_size, hidden_size, output_size, weight_init_std = 0.01)
        # 가중치 초기화
        self.params = {}
        self.params['W1'] = weight_init_std * np.random.randn(input_size, hidden_siz
        self.params['b1'] = np.zeros(hidden_size)
        self.params['W2'] = weight_init_std * np.random.randn(hidden_size, output_si
        self.params['b2'] = np.zeros(output_size)

        # 계층 생성
        self.layers = OrderedDict()
        self.layers['Affine1'] = Affine(self.params['W1'], self.params['b1'])
        self.layers['Relu1'] = Relu()
        self.layers['Affine2'] = Affine(self.params['W2'], self.params['b2'])

        self.lastLayer = SoftmaxWithLoss()

    def predict(self, x):
        for layer in self.layers.values():
            x = layer.forward(x)

        return x

    # x : 입력 데이터, t : 정답 레이블
    def loss(self, x, t):
        y = self.predict(x)
        return self.lastLayer.forward(y, t)

    def accuracy(self, x, t):
        y = self.predict(x)
        y = np.argmax(y, axis=1)
        if t.ndim != 1 : t = np.argmax(t, axis=1)

        accuracy = np.sum(y == t) / float(x.shape[0])
```

```python
def gradient(self, x, t):
    # forward
    self.loss(x, t)

    # backward
    dout = 1
    dout = self.lastLayer.backward(dout)

    layers = list(self.layers.values())
    layers.reverse()
    for layer in layers:
        dout = layer.backward(dout)

    # 결과 저장
    grads = {}
    grads['W1'], grads['b1'] = self.layers['Affine1'].dW, self.layers['Affine1'].db
    grads['W2'], grads['b2'] = self.layers['Affine2'].dW, self.layers['Affine2'].db

    return grads
```

Implementation of Two Layer NN – cont.

```python
# 데이터 읽기
(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, one_hot_label=True)

network = TwoLayerNet(input_size=784, hidden_size=50, output_size=10)

iters_num = 10000
train_size = x_train.shape[0]
batch_size = 100
learning_rate = 0.1

train_loss_list = []
train_acc_list = []
test_acc_list = []

iter_per_epoch = max(train_size / batch_size, 1)
```

```python
for i in range(iters_num):
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    t_batch = t_train[batch_mask]

    # 기울기 계산
    #grad = network.numerical_gradient(x_batch, t_batch) # 수치 미분 방식
    grad = network.gradient(x_batch, t_batch) # 오차역전파법 방식(훨씬 빠르다)

    # 갱신
    for key in ('W1', 'b1', 'W2', 'b2'):
        network.params[key] -= learning_rate * grad[key]

    loss = network.loss(x_batch, t_batch)
    train_loss_list.append(loss)

    if i % iter_per_epoch == 0:
        train_acc = network.accuracy(x_train, t_train)
        test_acc = network.accuracy(x_test, t_test)
        train_acc_list.append(train_acc)
        test_acc_list.append(test_acc)
        print(train_acc, test_acc)
```

수고하셨습니다.