Reading Circle #5

Chapter5 Backward Propagation Neural Network

2018/06/18

Zhongjie Zhang

D1

5.6 Realization of Affine and Softmax Layer

In the backward propagation network model,the propagation can be decomposed into several layers. The function of propagation is mainly rely on the affine layers.And in this example,we use softmax layer as the output layer.

5.6.1 Affine Layer

The parameters based on former node calculation are all numbers.

We apply Matrix multiplication to the calculation of the Neural Networks.

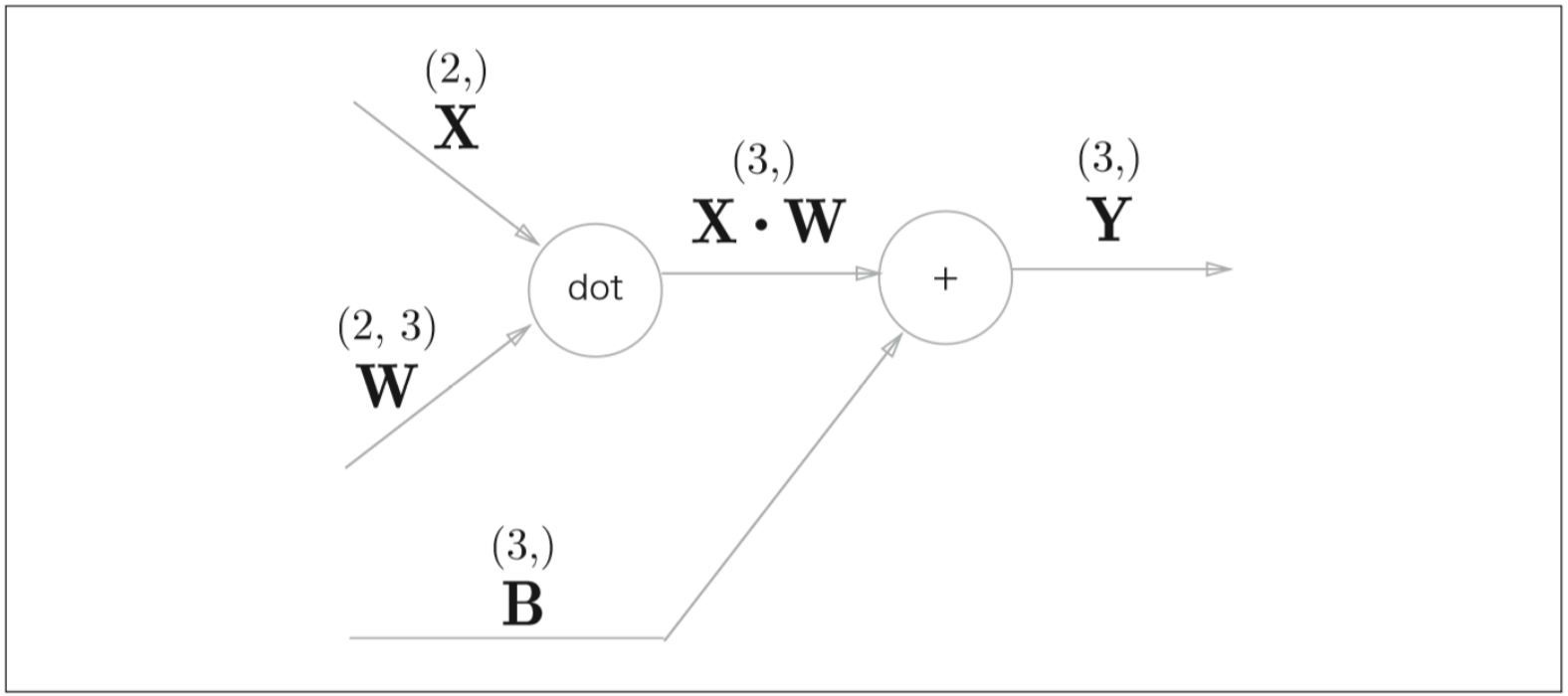
Y=np.dot(X,W)+B

So we can apply the Matrix multiplication to layer calculation and we call the layer as affine layer.

5.6.1 Affine Layer

We must pay attention to the rule of the Matrix multiplication that we must match the rows and columns.

X(1,2)*W(2,3)+b(1,3)=Y(1,3)



5.6.1 Affine Layer

Expression of Partial derivative

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

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

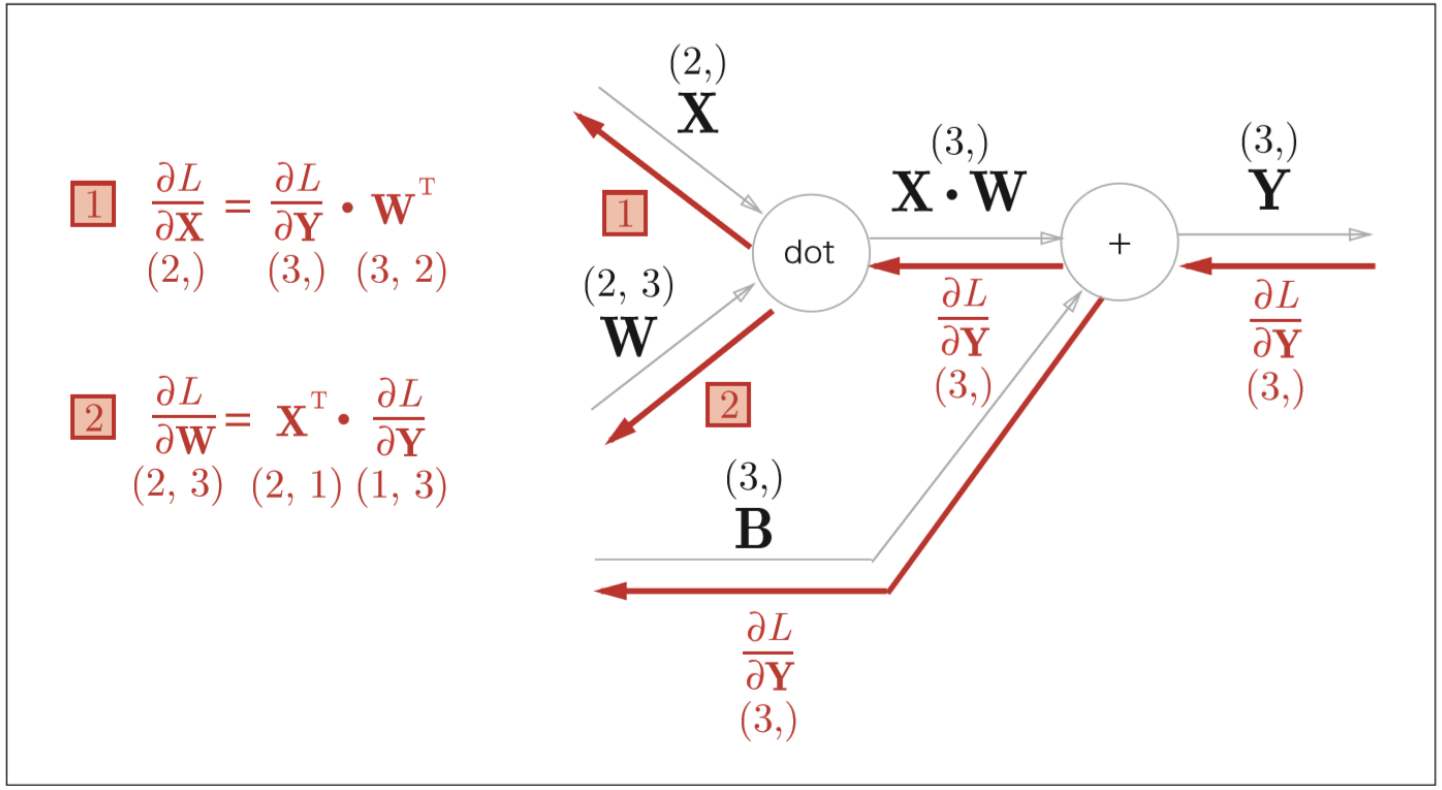
$$\mathbf{W} = \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{pmatrix}$$

$$\mathbf{W}^{\mathrm{T}} = \begin{pmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \\ w_{13} & w_{23} \end{pmatrix}$$

T: Matrix transpose

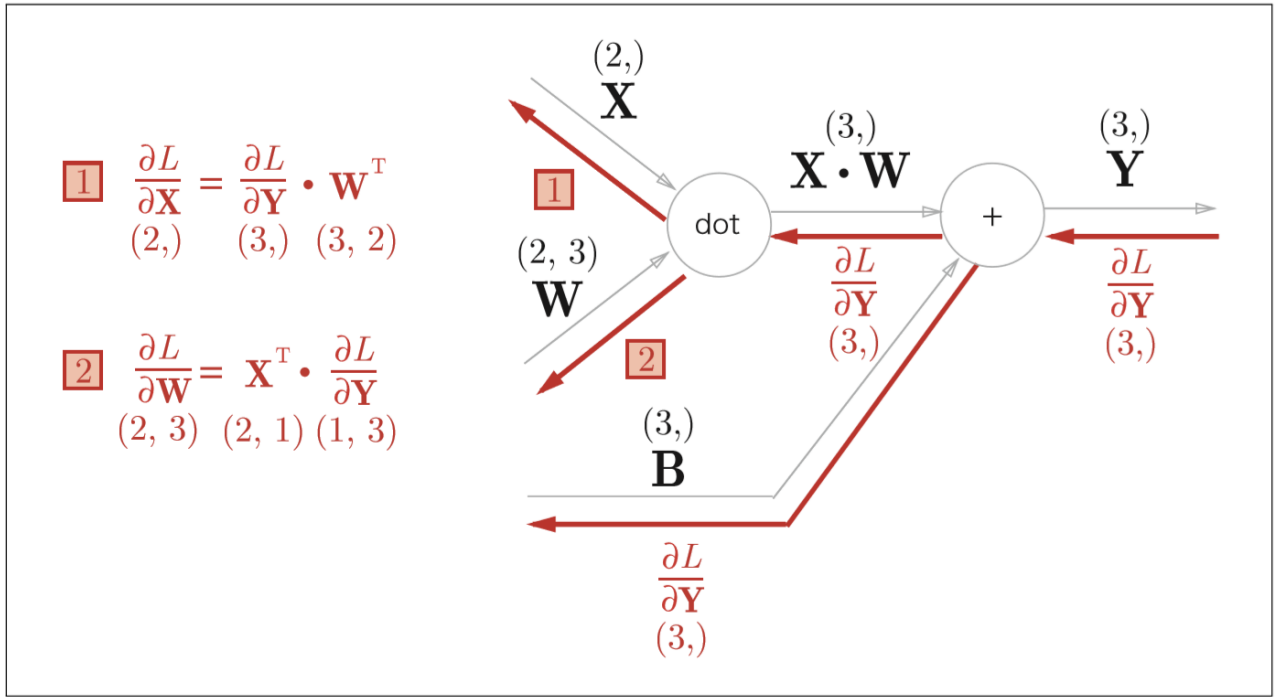
5.6.1 Affine Layer

We also need to pay attention to the rows and columns whether match or not in the backward propagation.



5.6.1 Affine Layer

Attention to shape

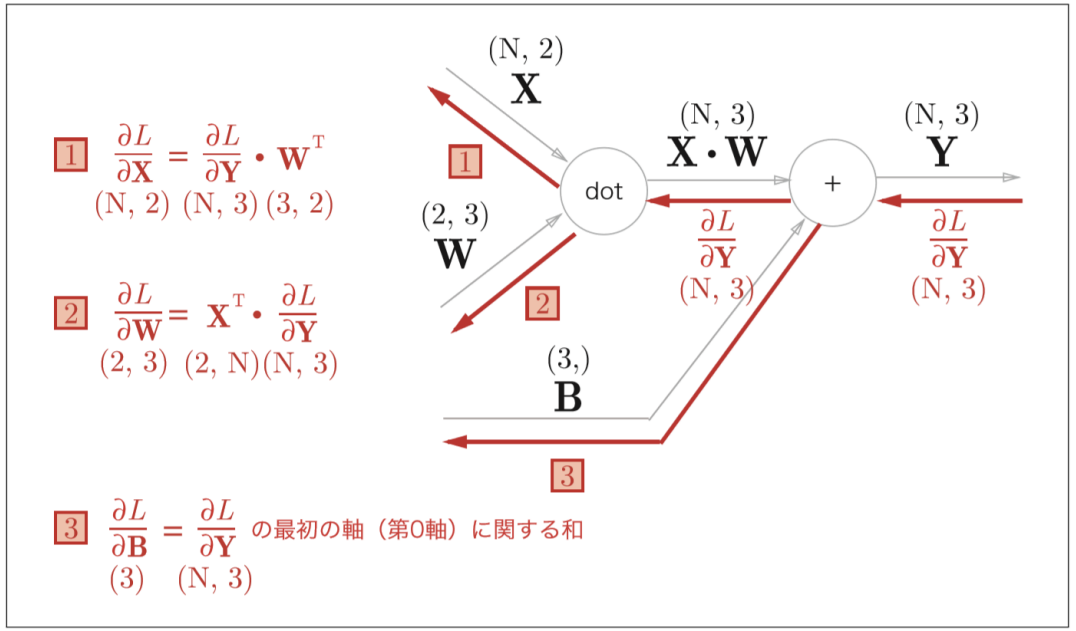


$$\mathbf{X} = (x_0, x_1, \cdots, x_n)$$

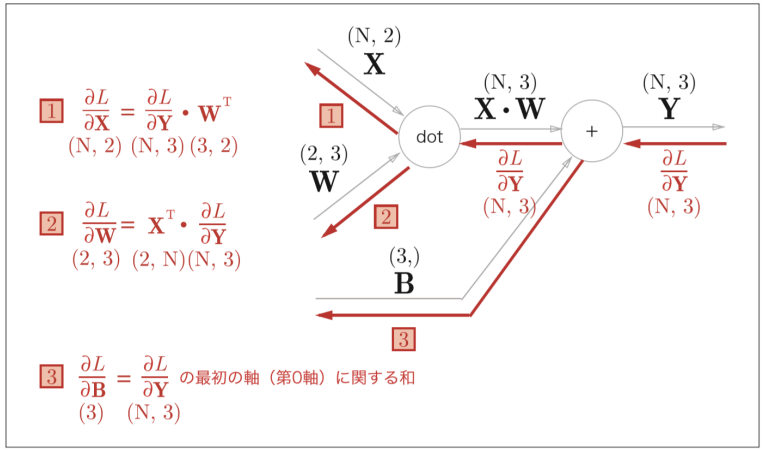
$$\frac{\partial L}{\partial \mathbf{X}} = \left(\frac{\partial L}{\partial x_0}, \frac{\partial L}{\partial x_1}, \cdots, \frac{\partial L}{\partial x_n} \right)$$

5.6.2 Batch Affine Layer

The principle is like Chapter 4. Assuming the data is a (1,M)Matrix,then the (N*M)matrix means that there are N datas input at the same time.This batch process can improve the efficiency of the calculation.



5.6.2 Batch Affine Layer



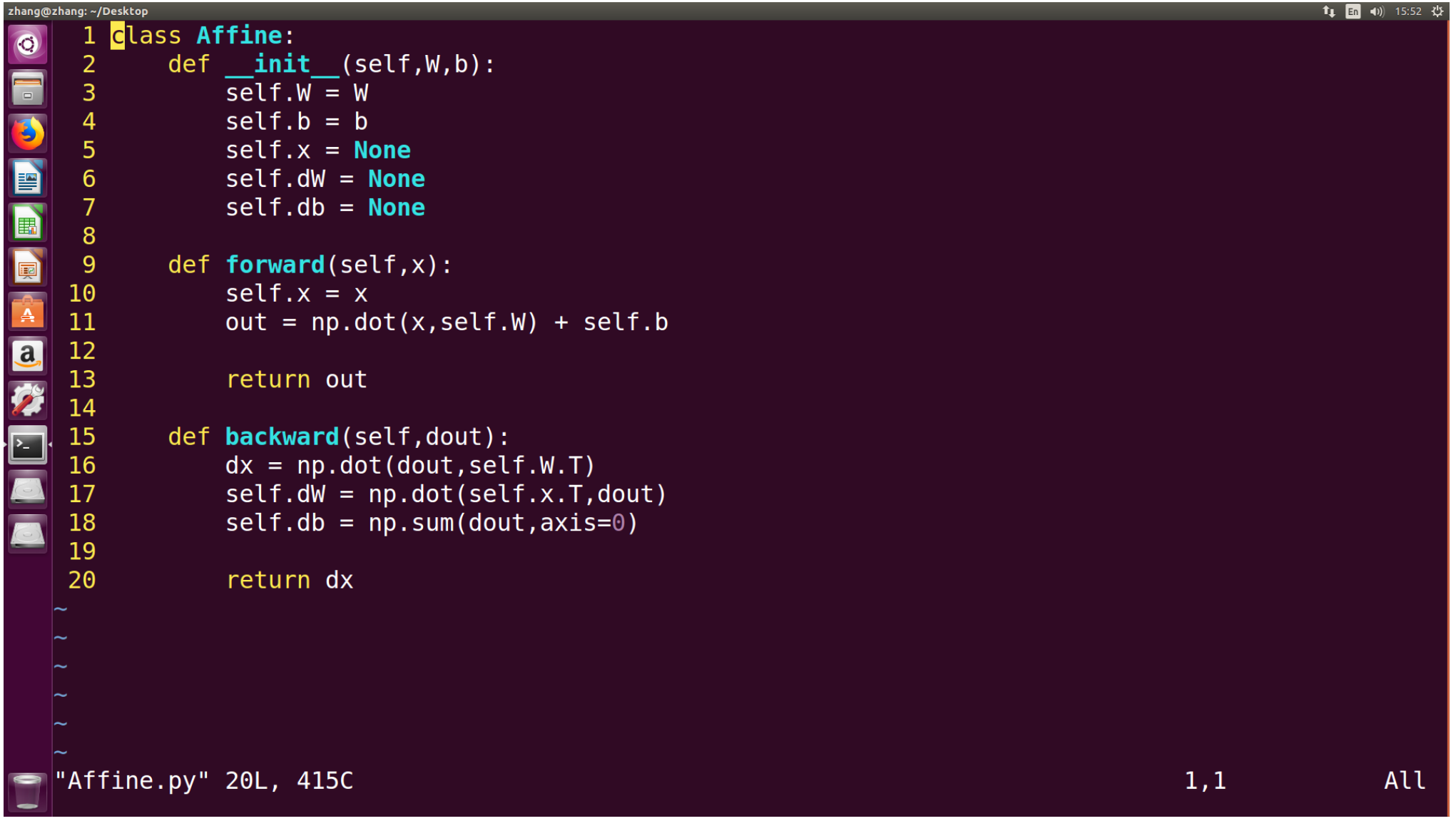
About the bias

We add the bias to all the data in the forward propagation.So we need to add all the bias in rows.

```
>>> X_dot_W = np.array([[0, 0, 0], [10, 10, 10]])
>>> B = np.array([1, 2, 3])
>>>
>>> X_dot_W
array([[ 0,  0,  0],
       [ 10, 10, 10]])
>>> X_dot_W + B
array([[ 1,  2,  3],
       [11, 12, 13]])
```

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

5.6.2 Batch Affine Layer



5.6.3 Softmax-with-Loss Layer

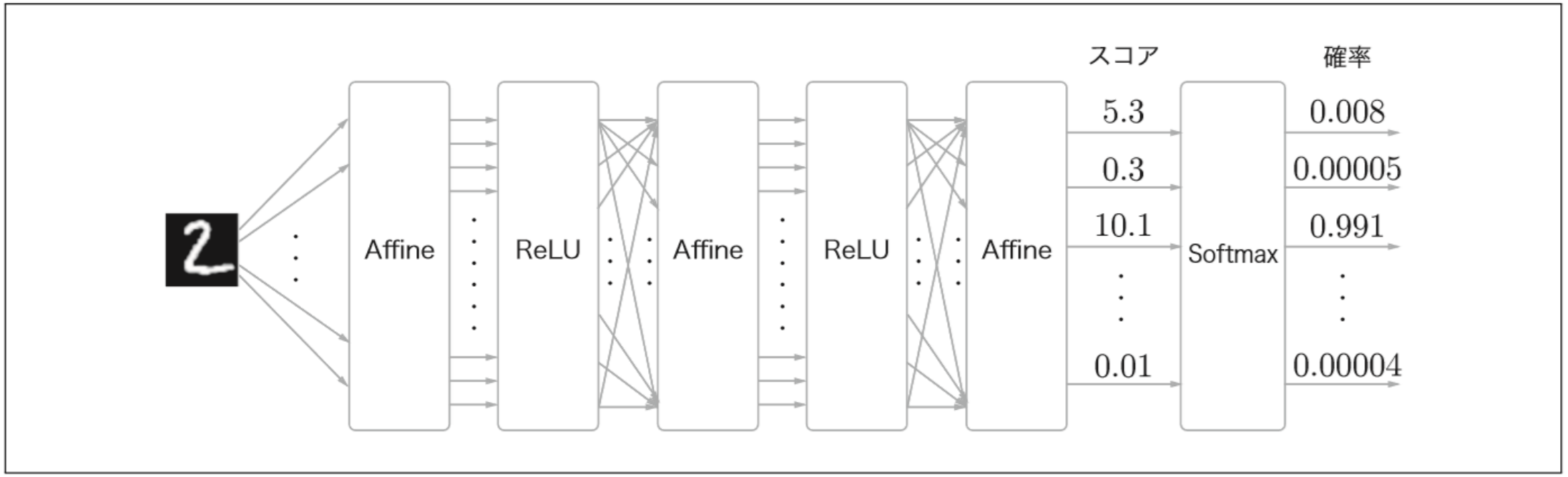
Softmax Function can make the value of the output normalized which means the sum of all the output's value is 1.

According to the monotonicity of the softmax function,we can use the value of output before the softmax to do the inference.

And we call the value as score.

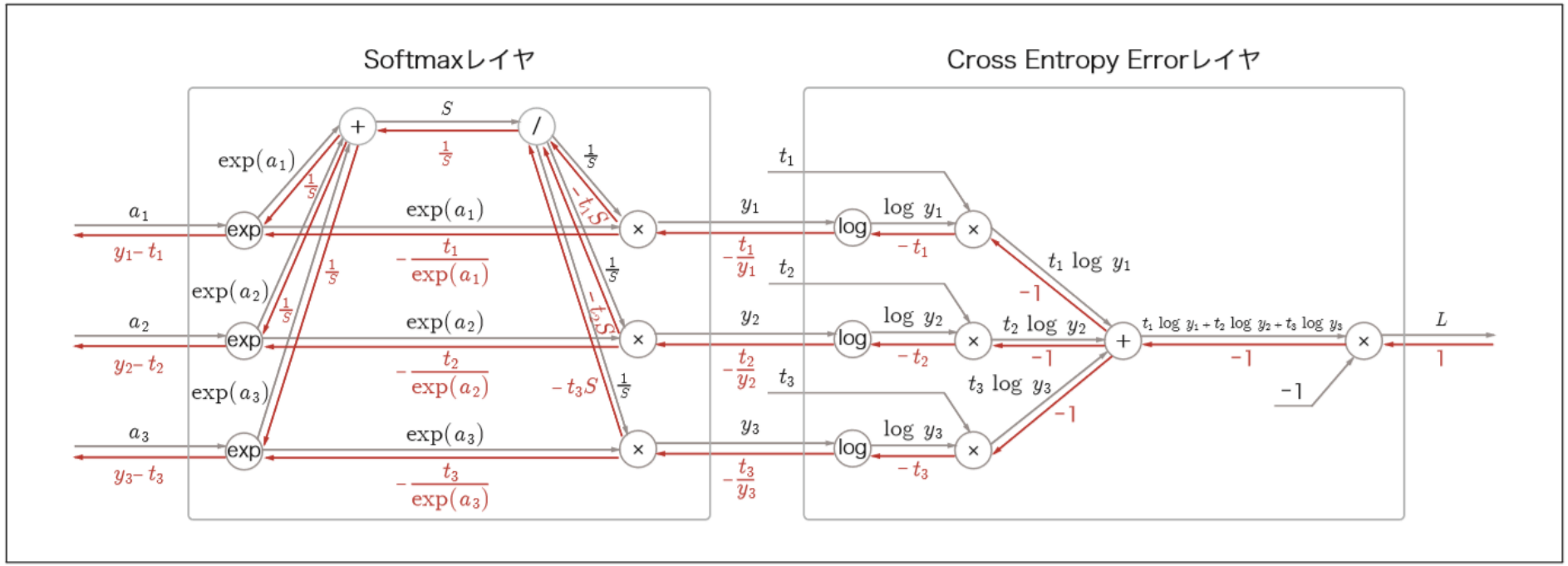
But if we need the data to do the learning,we should use the softmax function to make all the value normalized.

5.6.3 Softmax-with-Loss Layer



5.6.3 Softmax-with-Loss Layer

In order to make learning more easier,we put the output function and loss function together.We use softmax as output and cross entropy error as loss function.



5.6.3 Softmax-with-Loss Layer

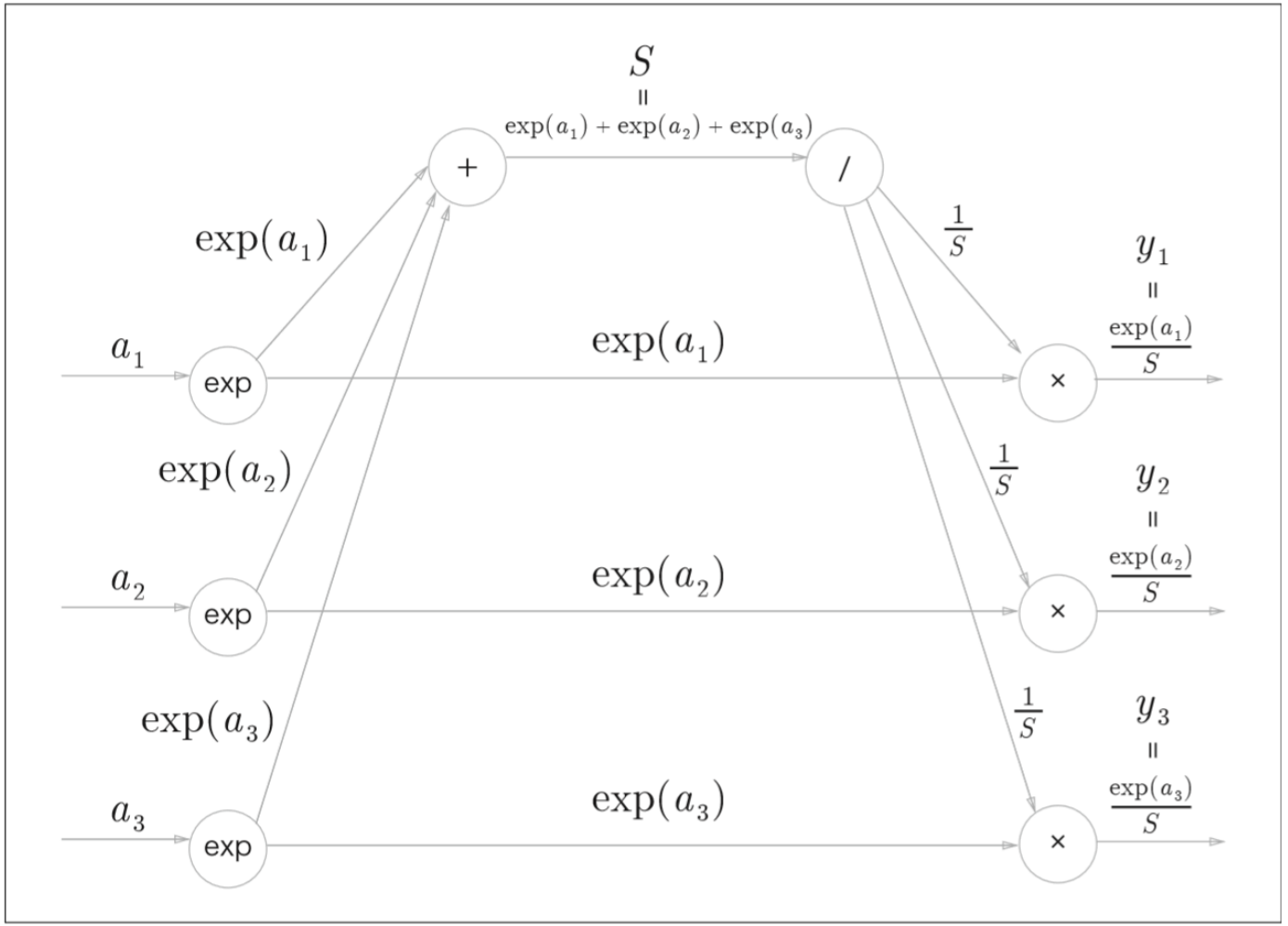
Forward Propagation

$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n} \exp(a_i)}$$

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

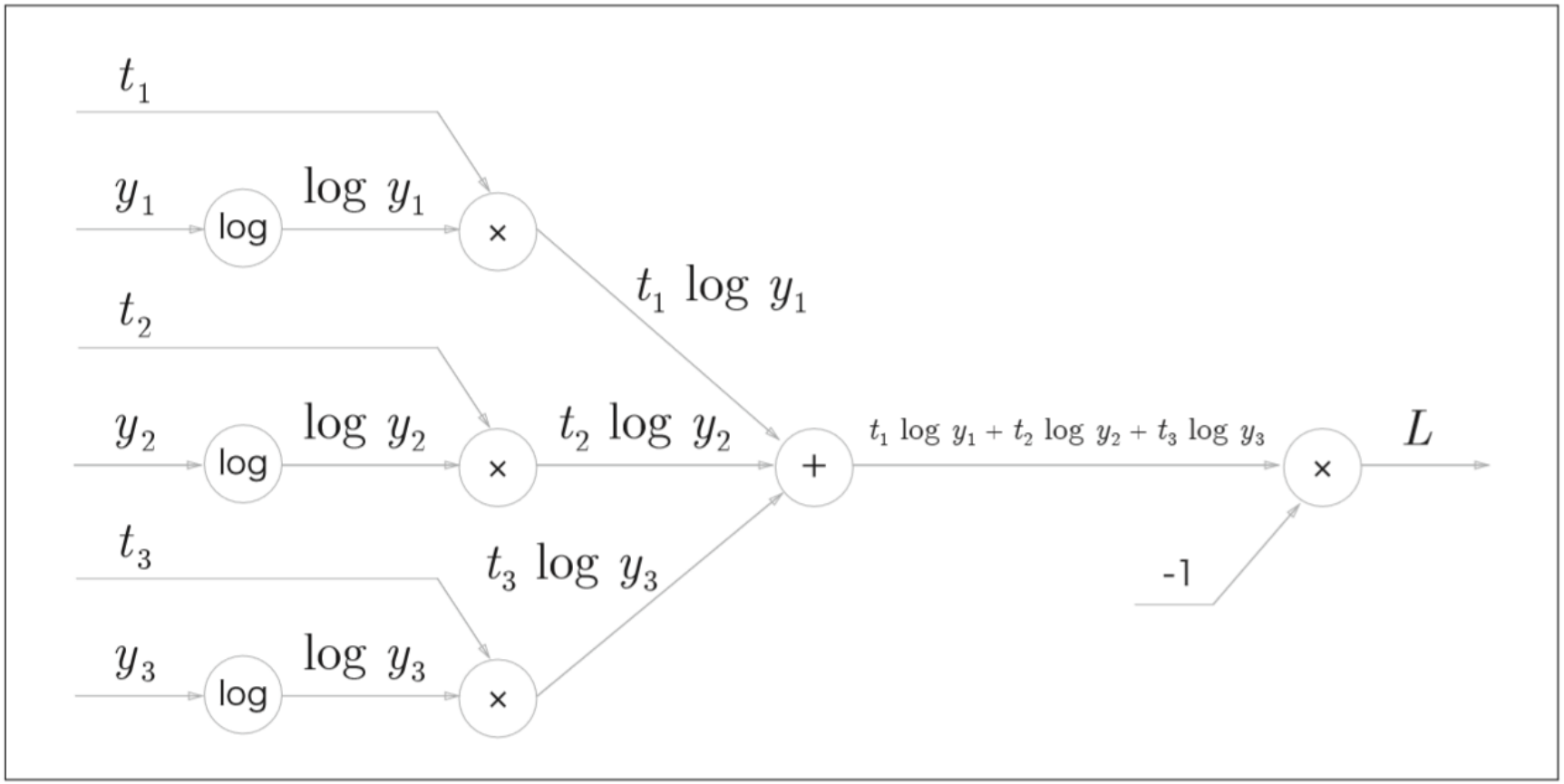
5.6.3 Softmax-with-Loss Layer

Forward Propagation



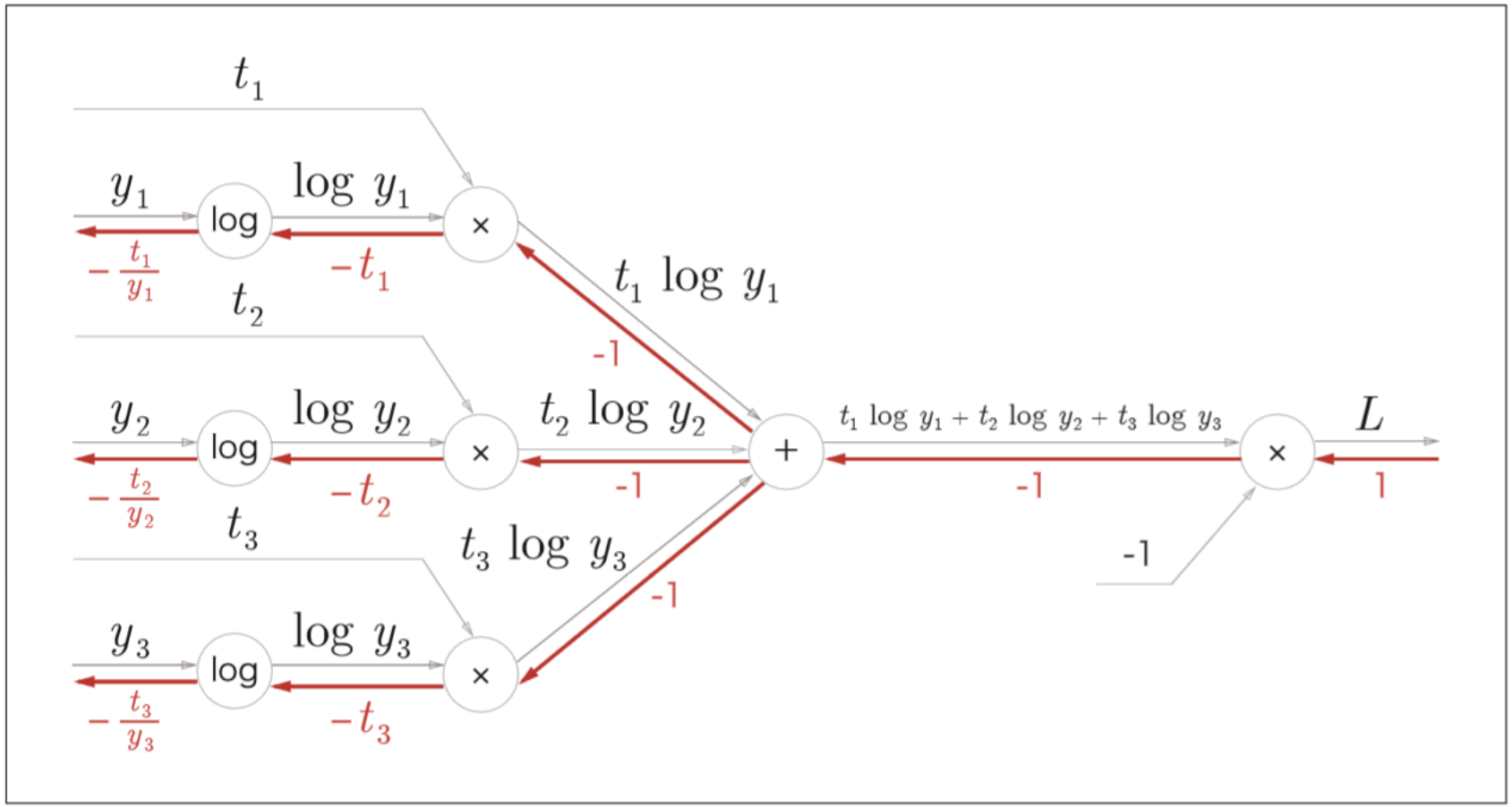
5.6.3 Softmax-with-Loss Layer

Forward Propagation



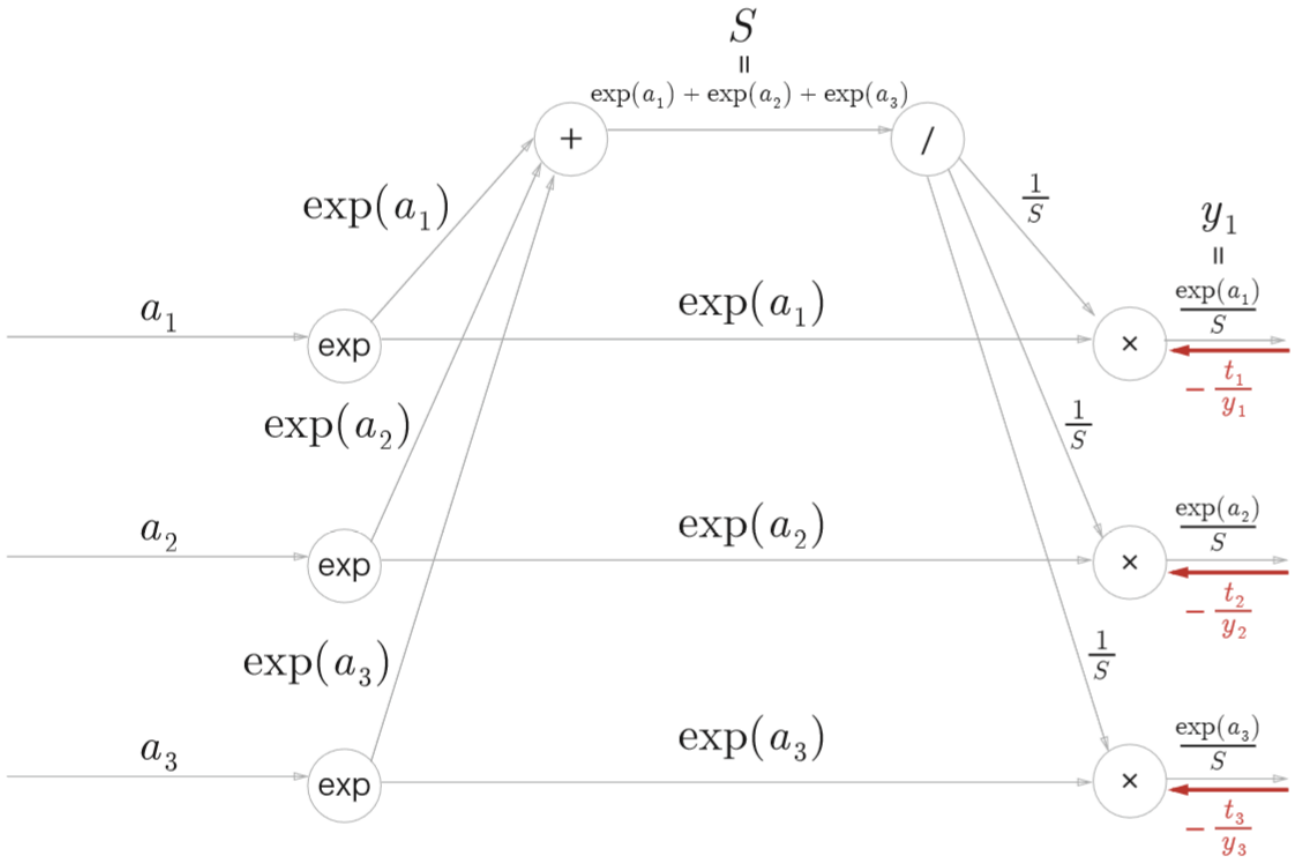
5.6.3 Softmax-with-Loss Layer

Backward propagation



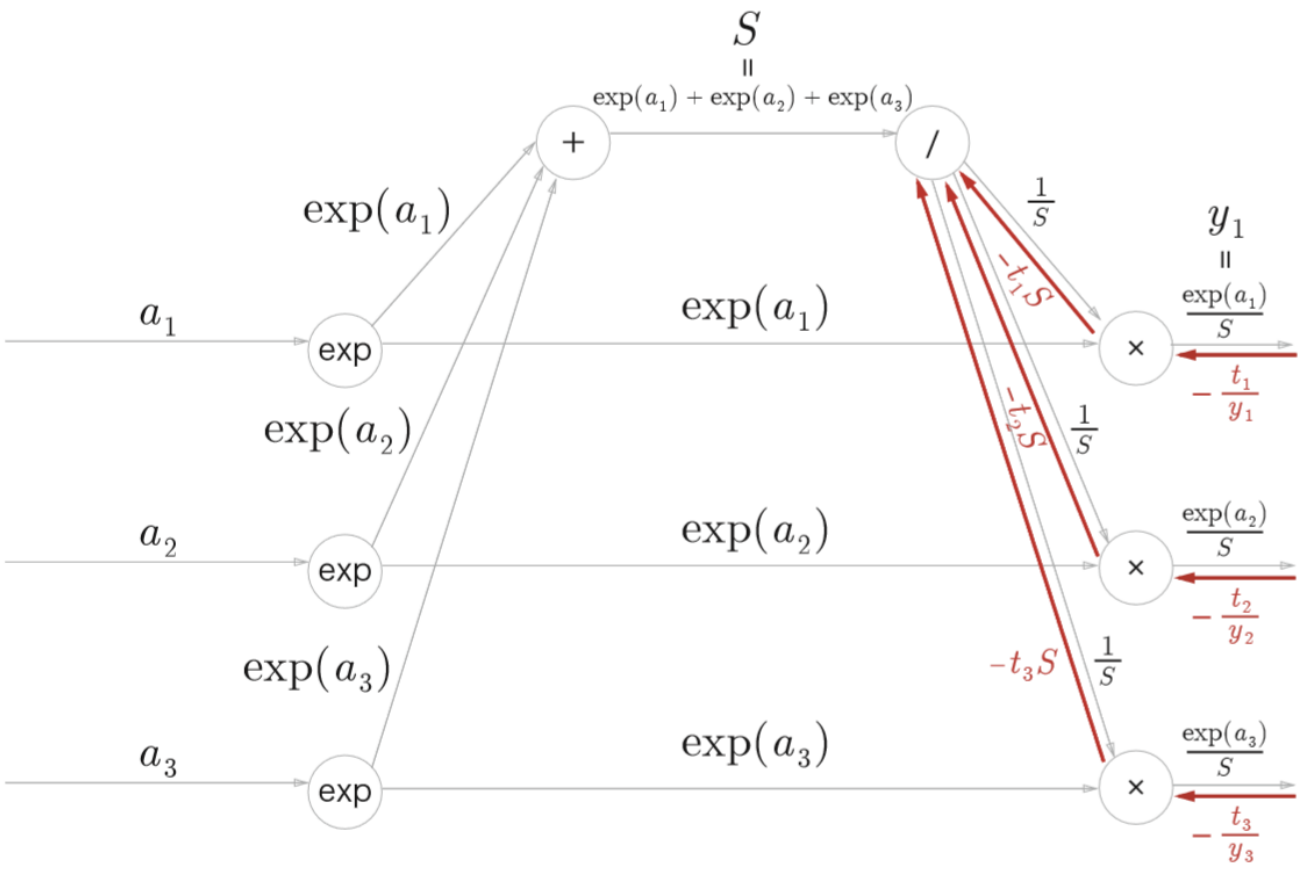
5.6.3 Softmax-with-Loss Layer

Backward propagation



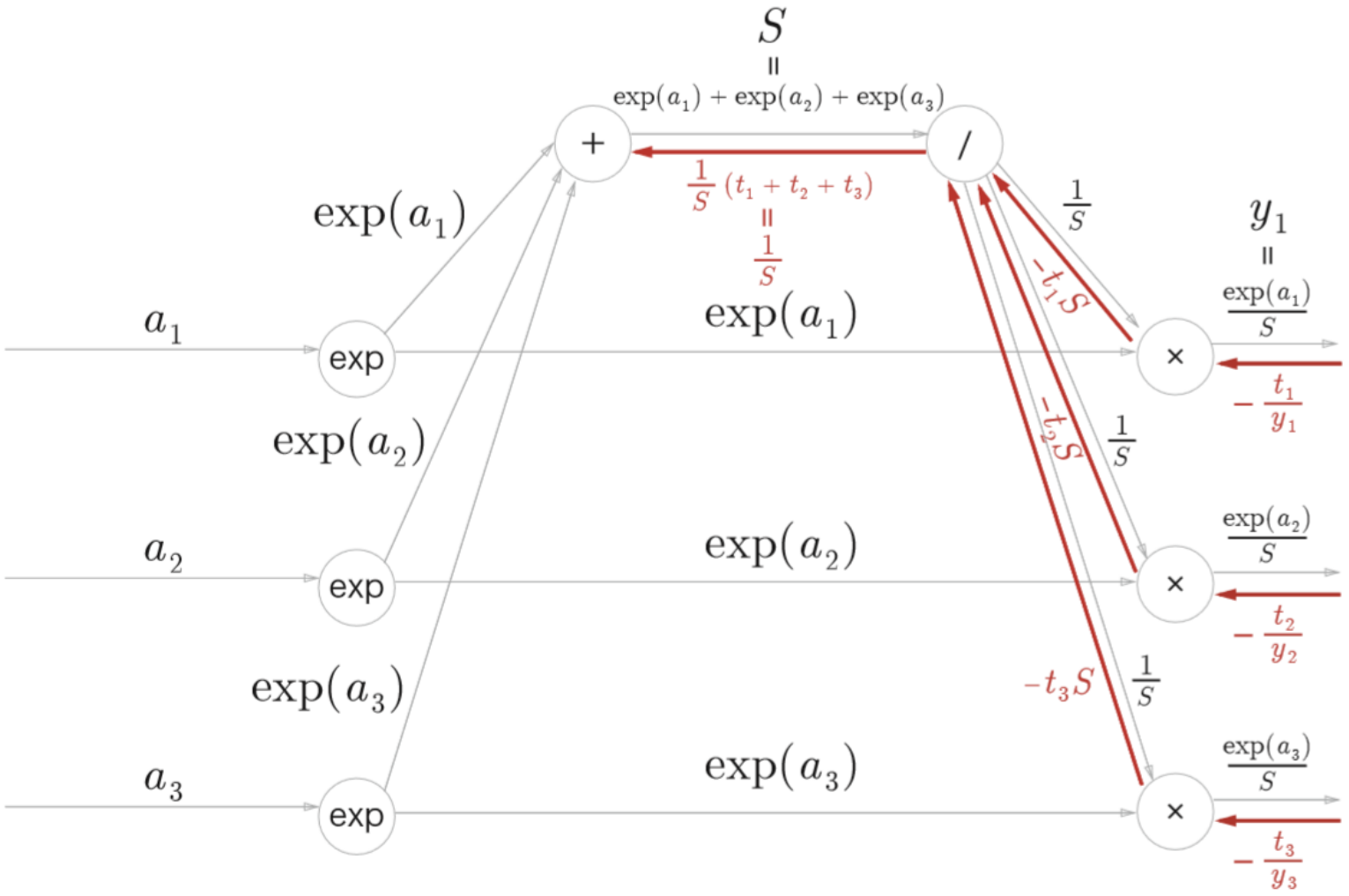
5.6.3 Softmax-with-Loss Layer

Backward propagation



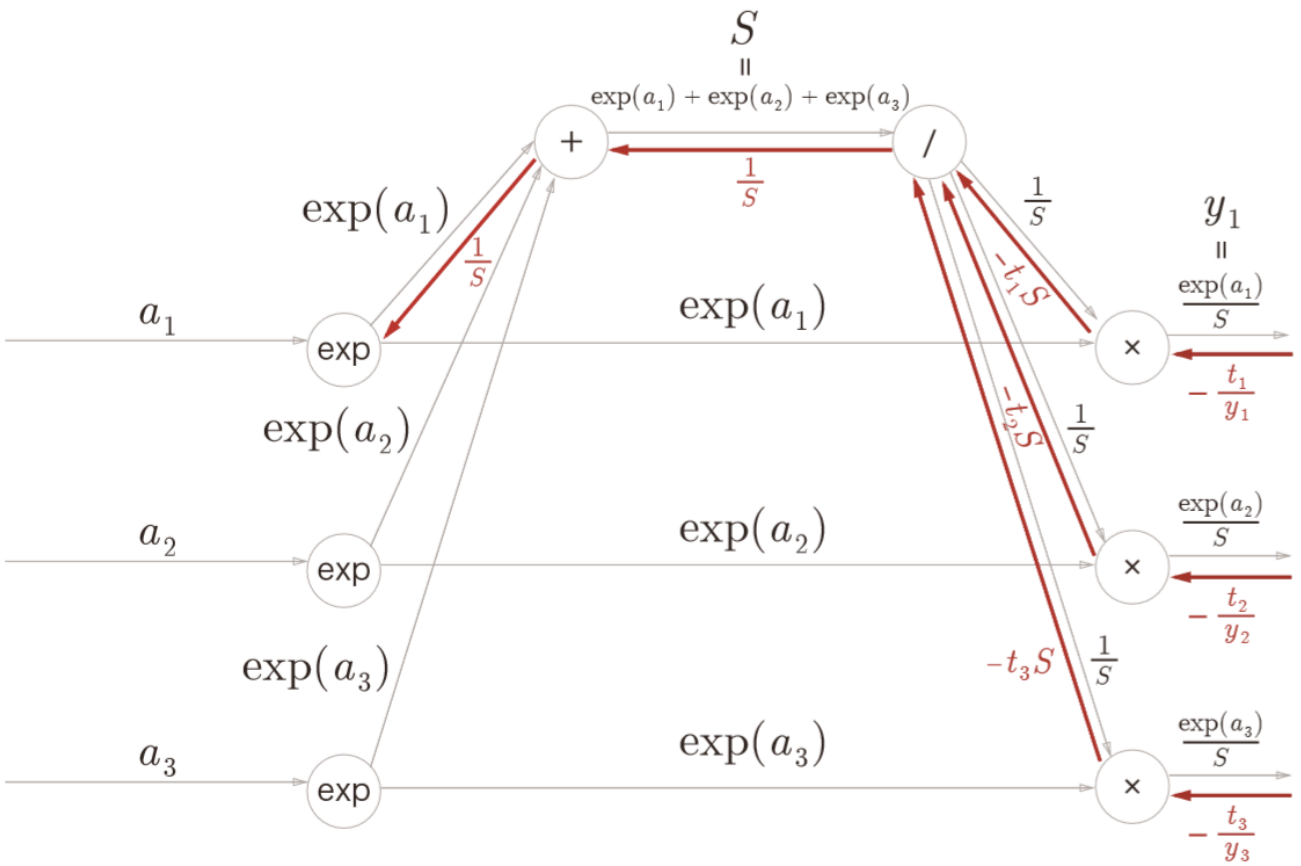
5.6.3 Softmax-with-Loss Layer

Backward propaga



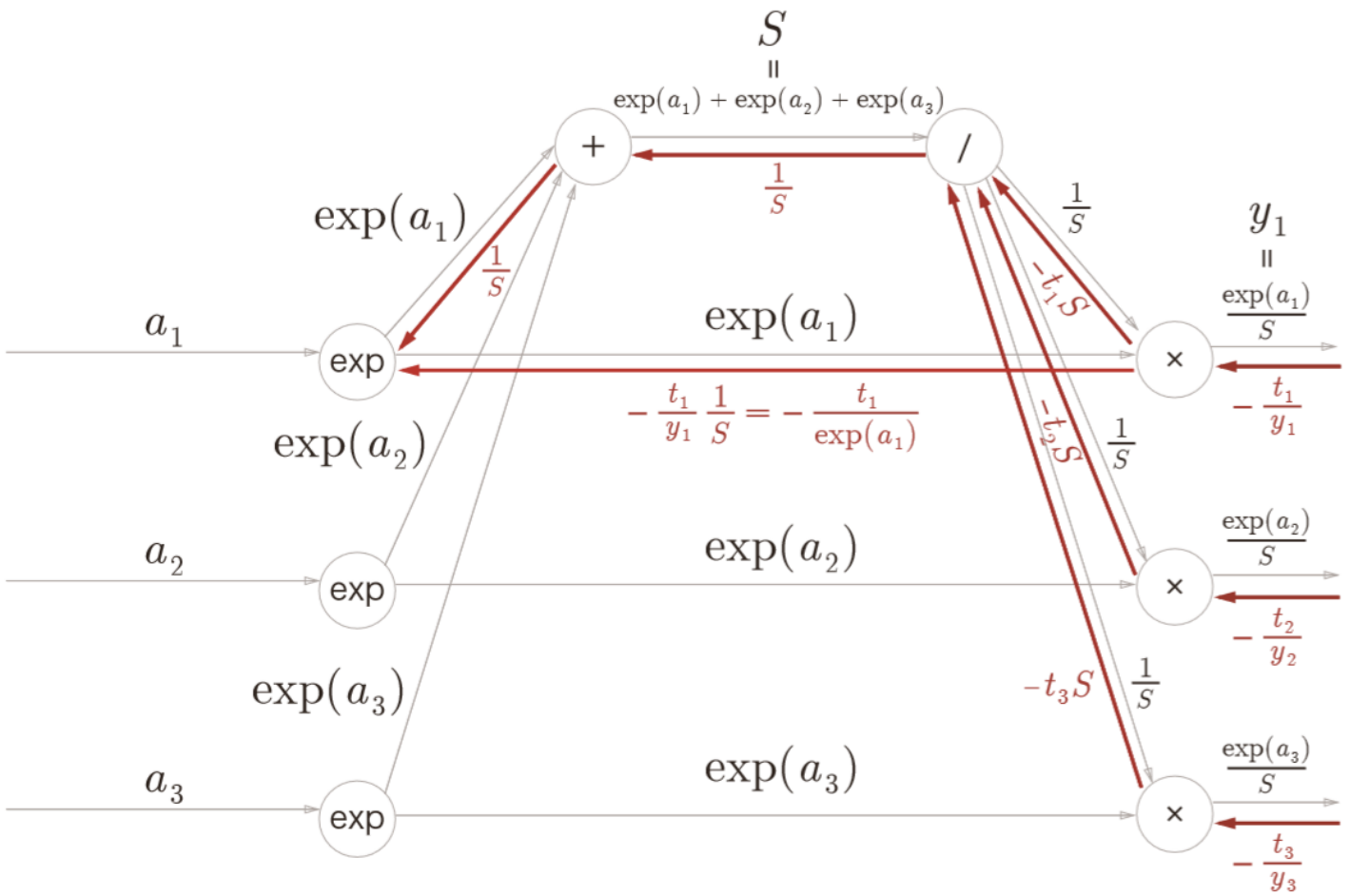
5.6.3 Softmax-with-Loss Layer

Backward propagatior



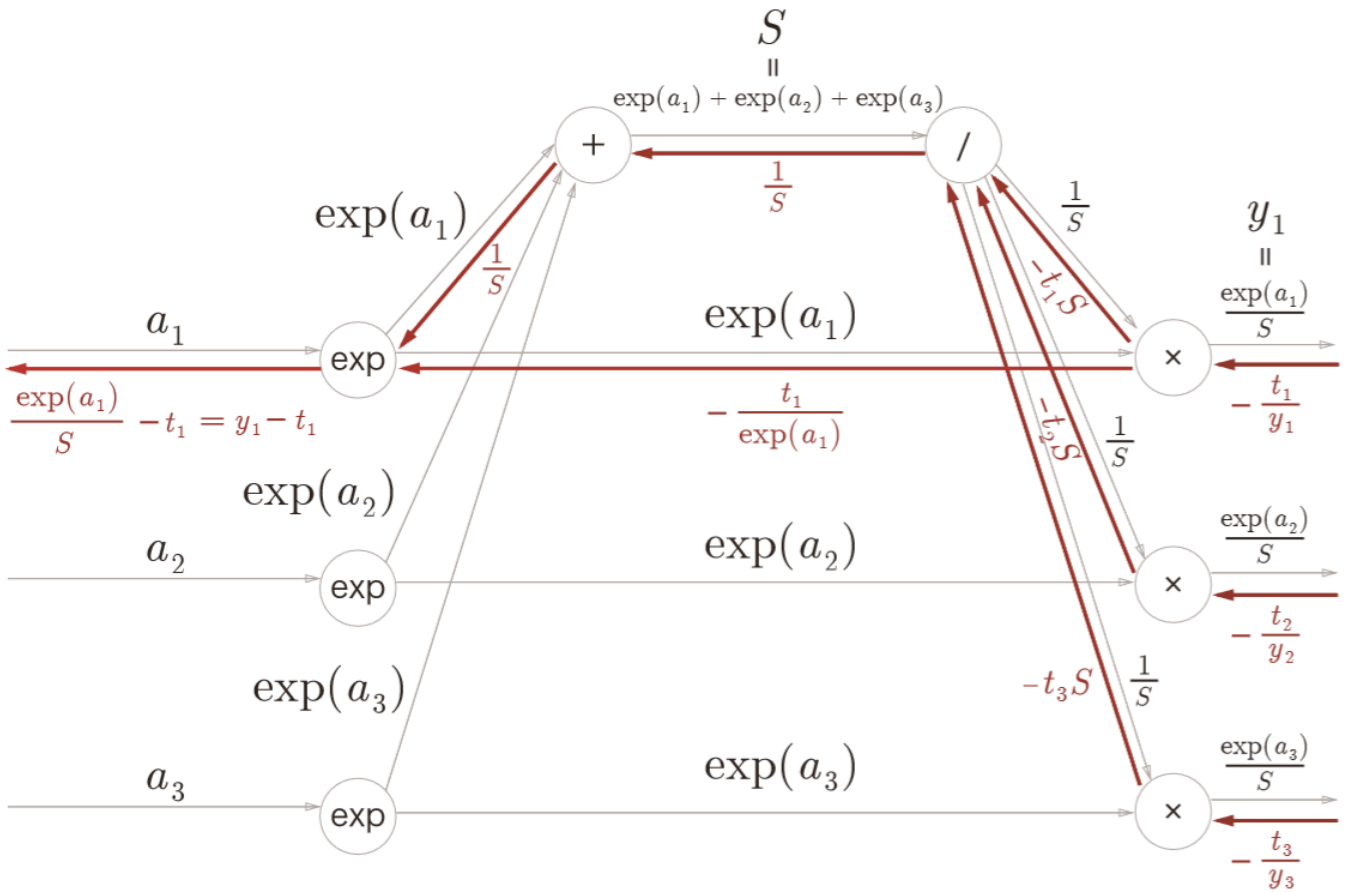
5.6.3 Softmax-with-Loss Layer

Backward propaga



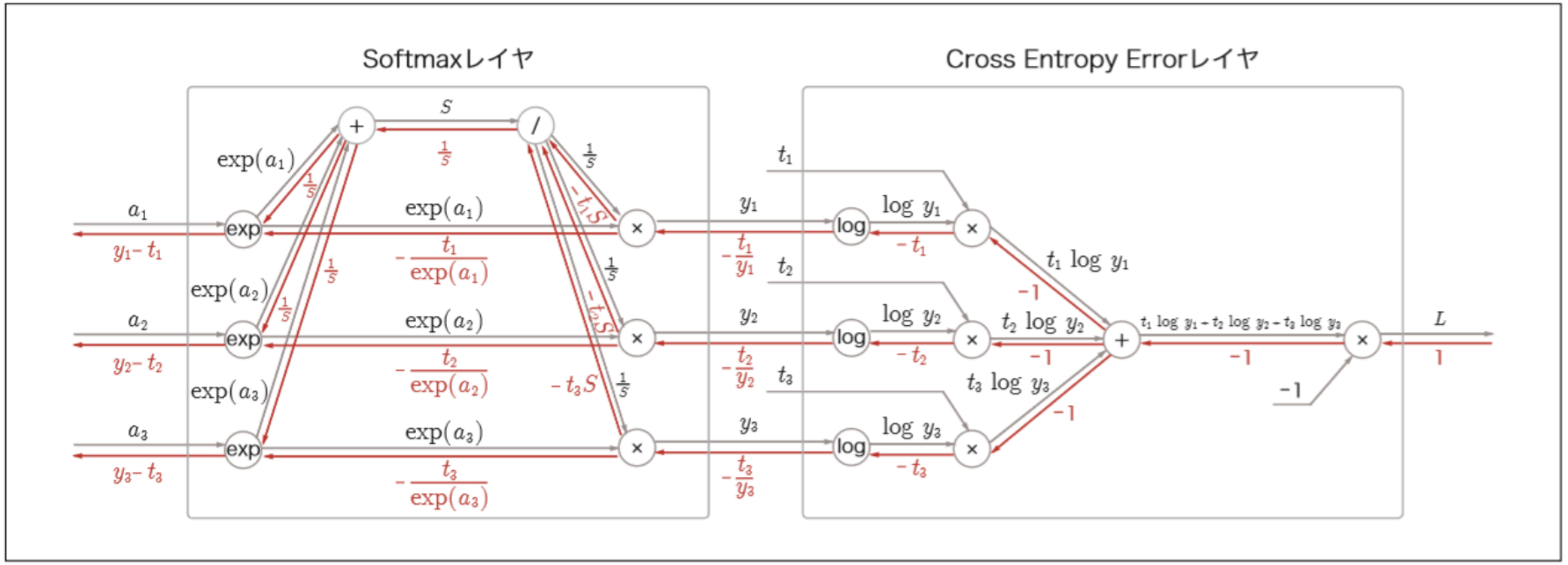
5.6.3 Softmax-with-Loss Layer

Backward propagation



5.6.3 Softmax-with-Loss Layer

Partial derivative = y_i - t_i



5.6.3 Softmax-with-Loss Layer

Simplification

$y_i - t_i =$

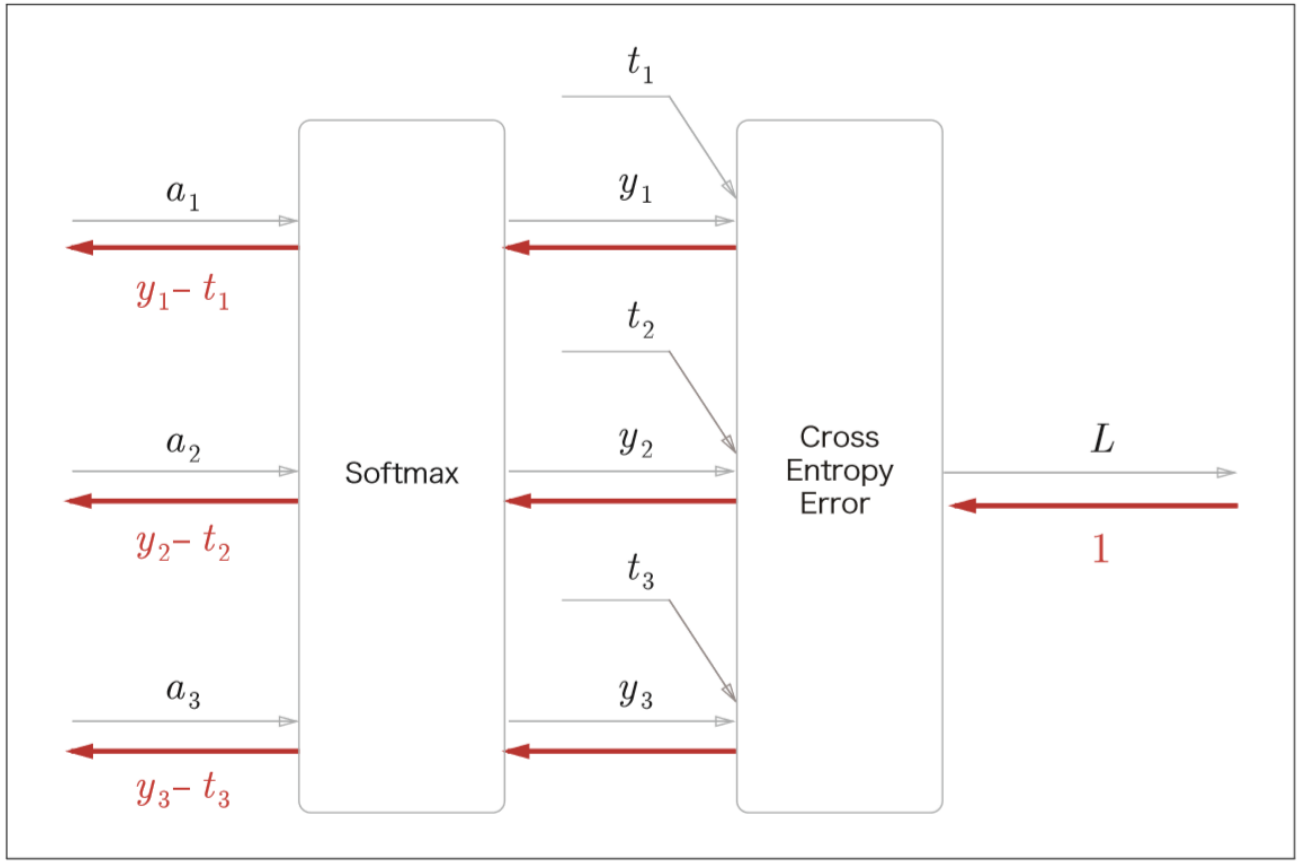
(Softmax +

Cross Entropy Error)

Or

(Identity function +

Mean squared error)



5.6.3 Softmax-with-Loss Layer

As seen above,we can decide how much effort we should take due to the value from the backward propagation.

For example,if the t_label is(0,1,0),the output value from softmax is(0.3,0.2,0.5).Then the value is(0.3,-0.8,0.5).That means we need a high level learning.

If the output value from softmax is(0.01,0.99,0),thus the value is(0.01,-0.01,0).That means we just need a low level learning.

5.6.3 Softmax-with-Loss Layer

5.7 realization of the back propagation

We can build the neural network based on the description and introduction of layers we just discussed.

5.7.1 overview of the network's learning

Learning means that we can automatically get the suitable weight and bias from the training data.

Here are the steps:

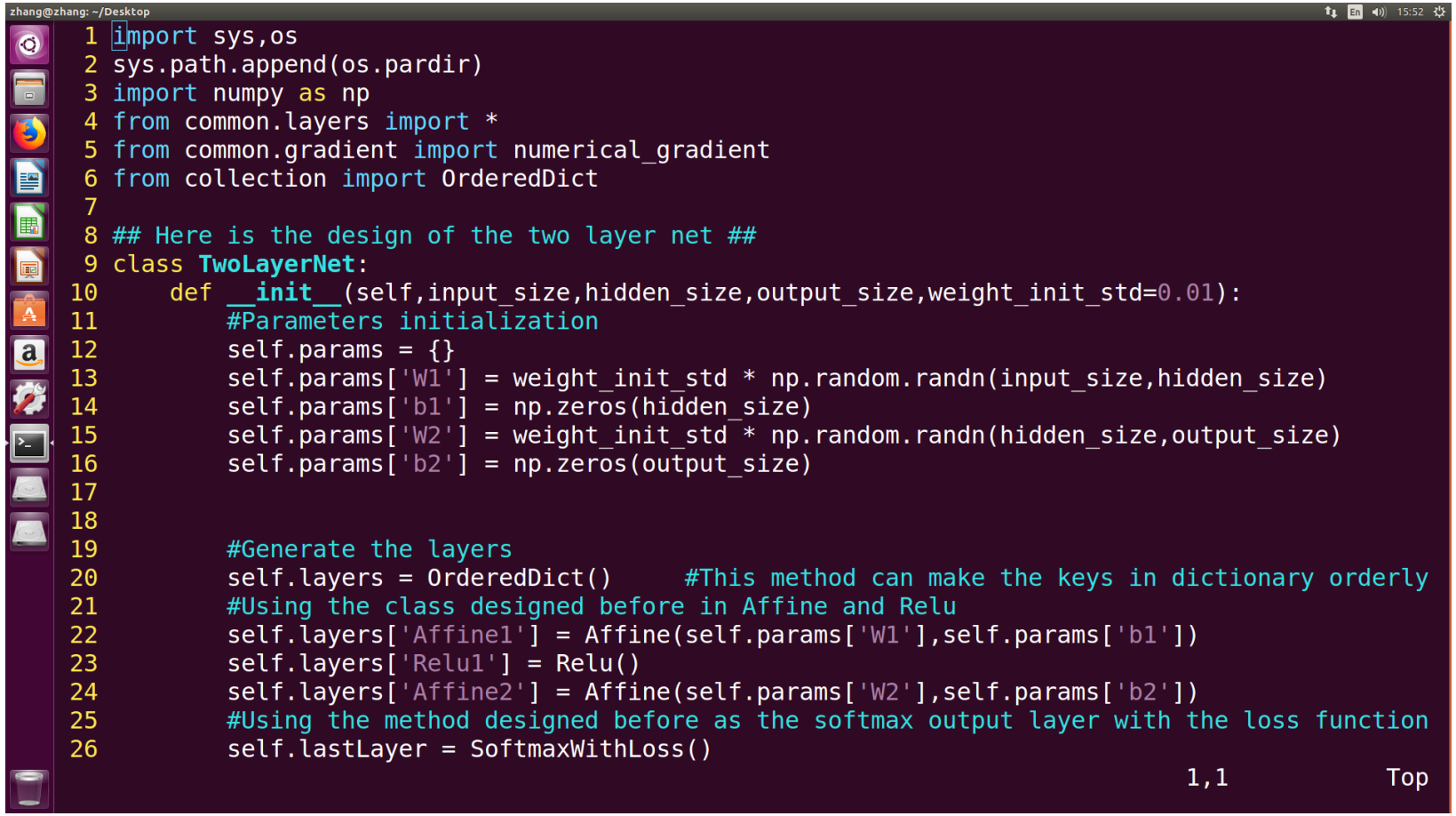
1.Choose the data randomly from the training data.

2.Calculate the gradient of the loss function.

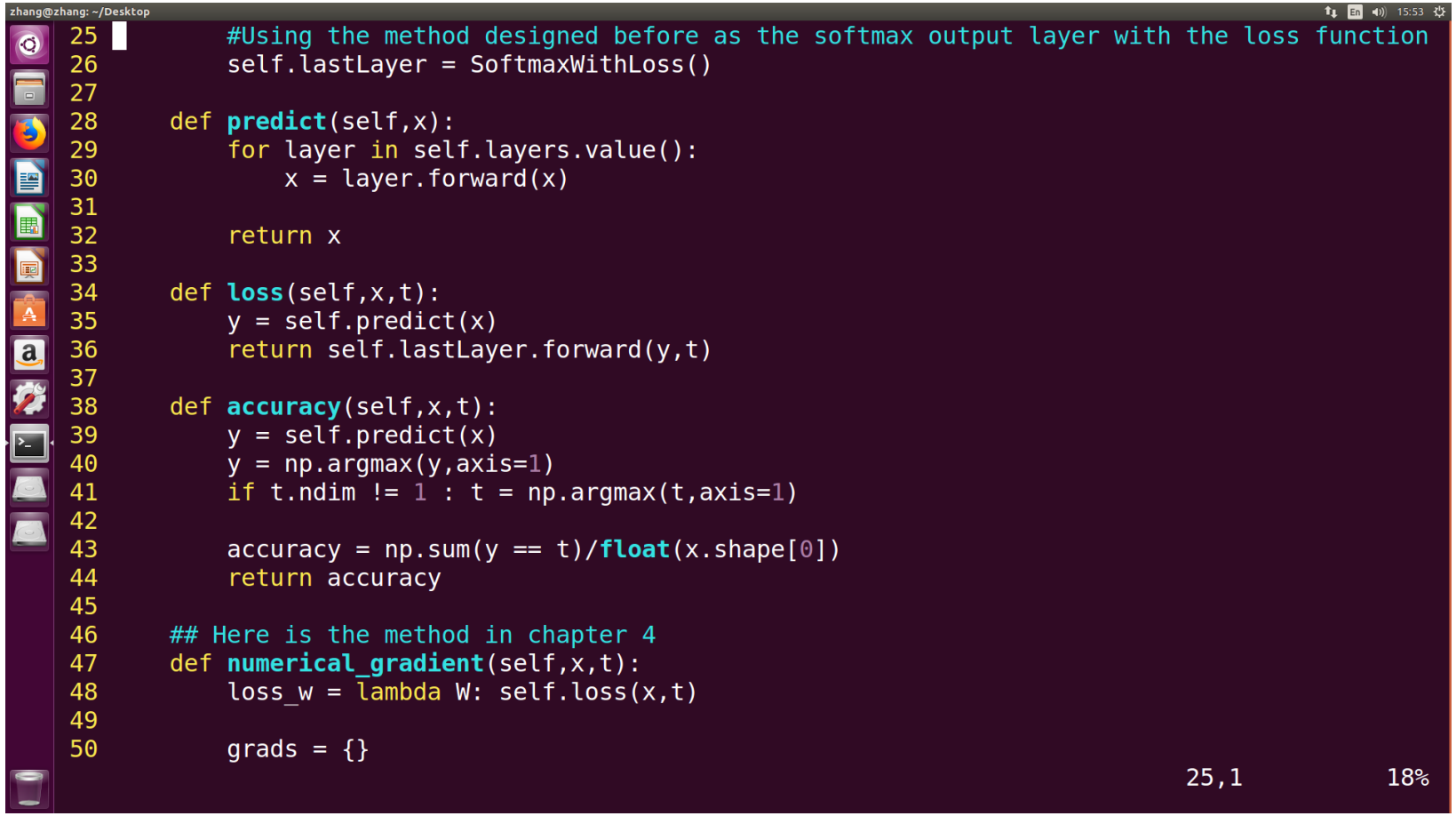
3.Change the parameters a little along the gradient direction

4. Repeat the 3 steps above

5.7.2 realization of backward propagation network

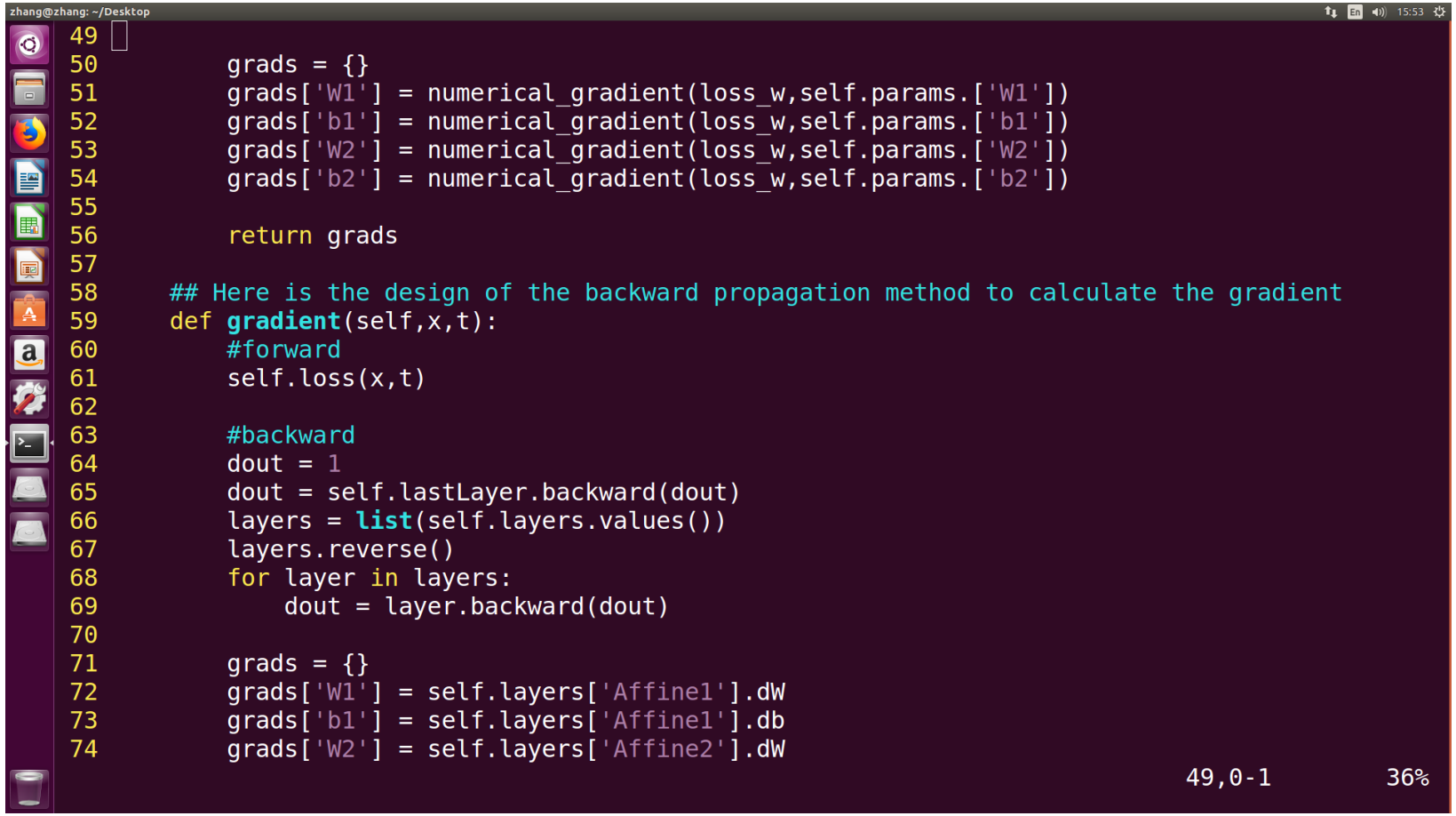


5.7.2 realization of backward propagation network

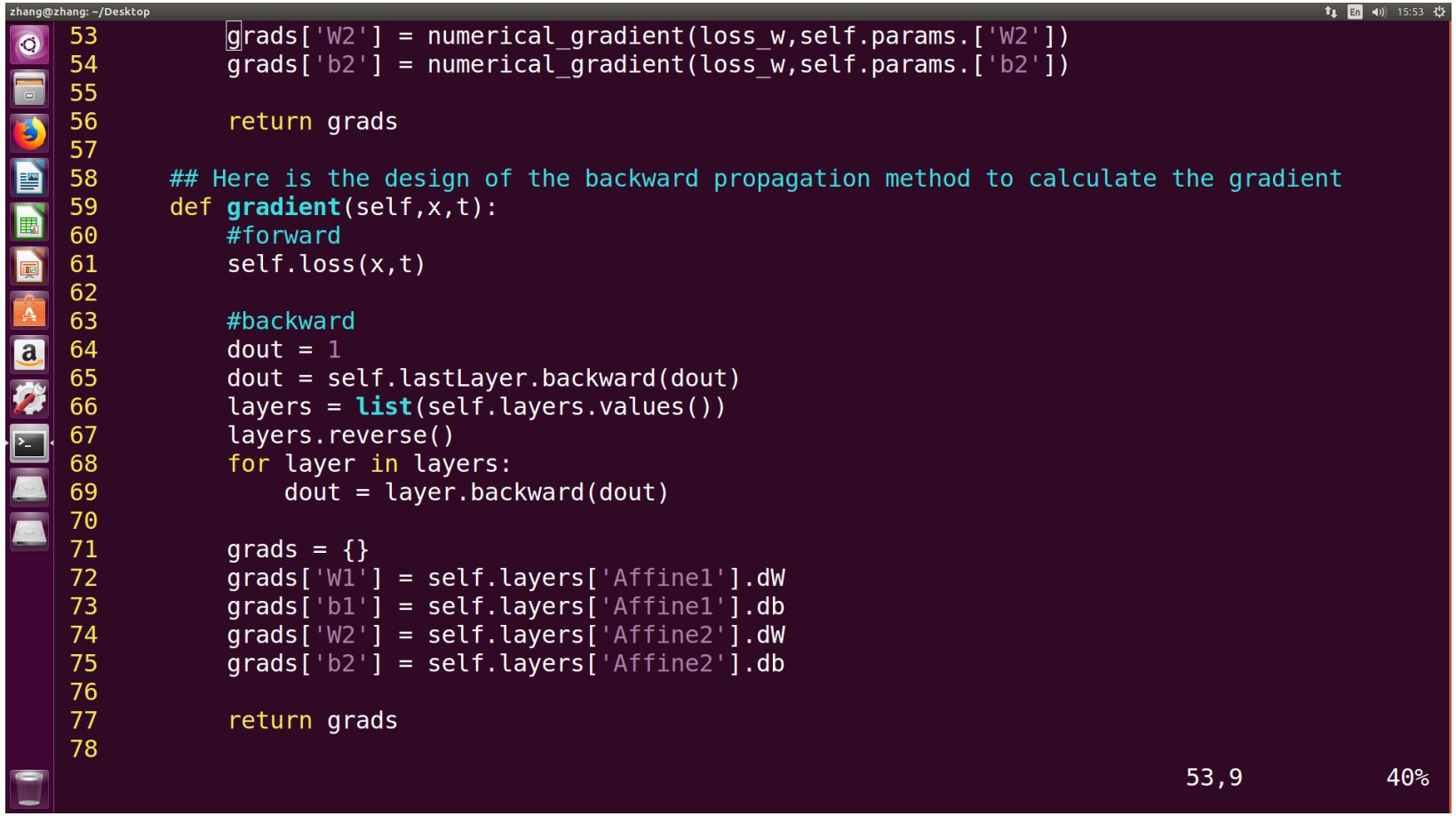



```
25          #Using the method designed before as the softmax output layer with the loss function
26          self.lastLayer = SoftmaxWithLoss()
27
28      def predict(self,x):
29          for layer in self.layers.value():
30              x = layer.forward(x)
31
32          return x
33
34      def loss(self,x,t):
35          y = self.predict(x)
36          return self.lastLayer.forward(y,t)
37
38      def accuracy(self,x,t):
39          y = self.predict(x)
40          y = np.argmax(y,axis=1)
41          if t.ndim != 1 : t = np.argmax(t,axis=1)
42
43          accuracy = np.sum(y == t)/float(x.shape[0])
44          return accuracy
45
46      ## Here is the method in chapter 4
47      def numerical_gradient(self,x,t):
48          loss_w = lambda W: self.loss(x,t)
49
50          grads = {}
```

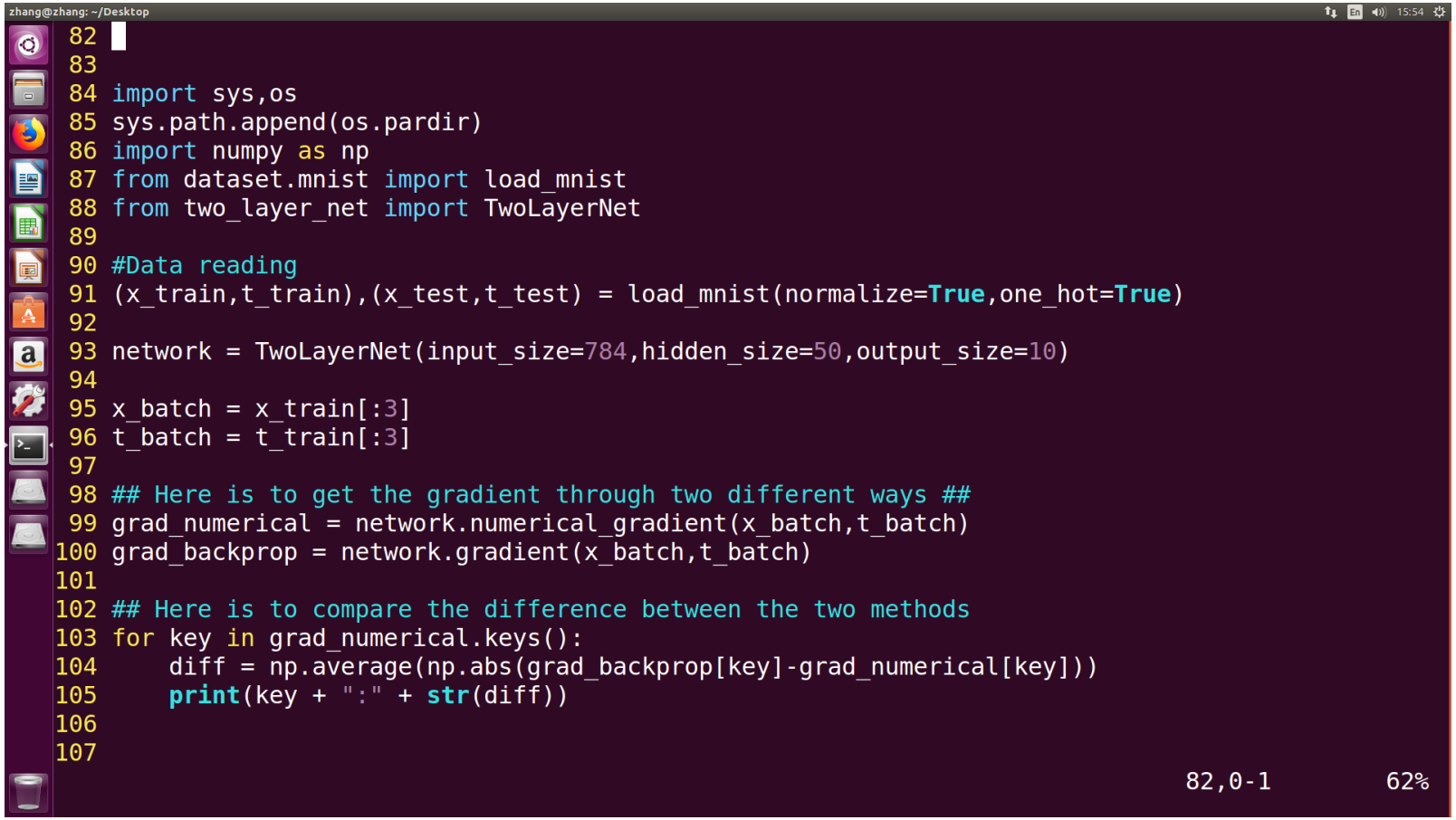
5.7.2 realization of backward propagation network



5.7.2 realization of backward propagation network

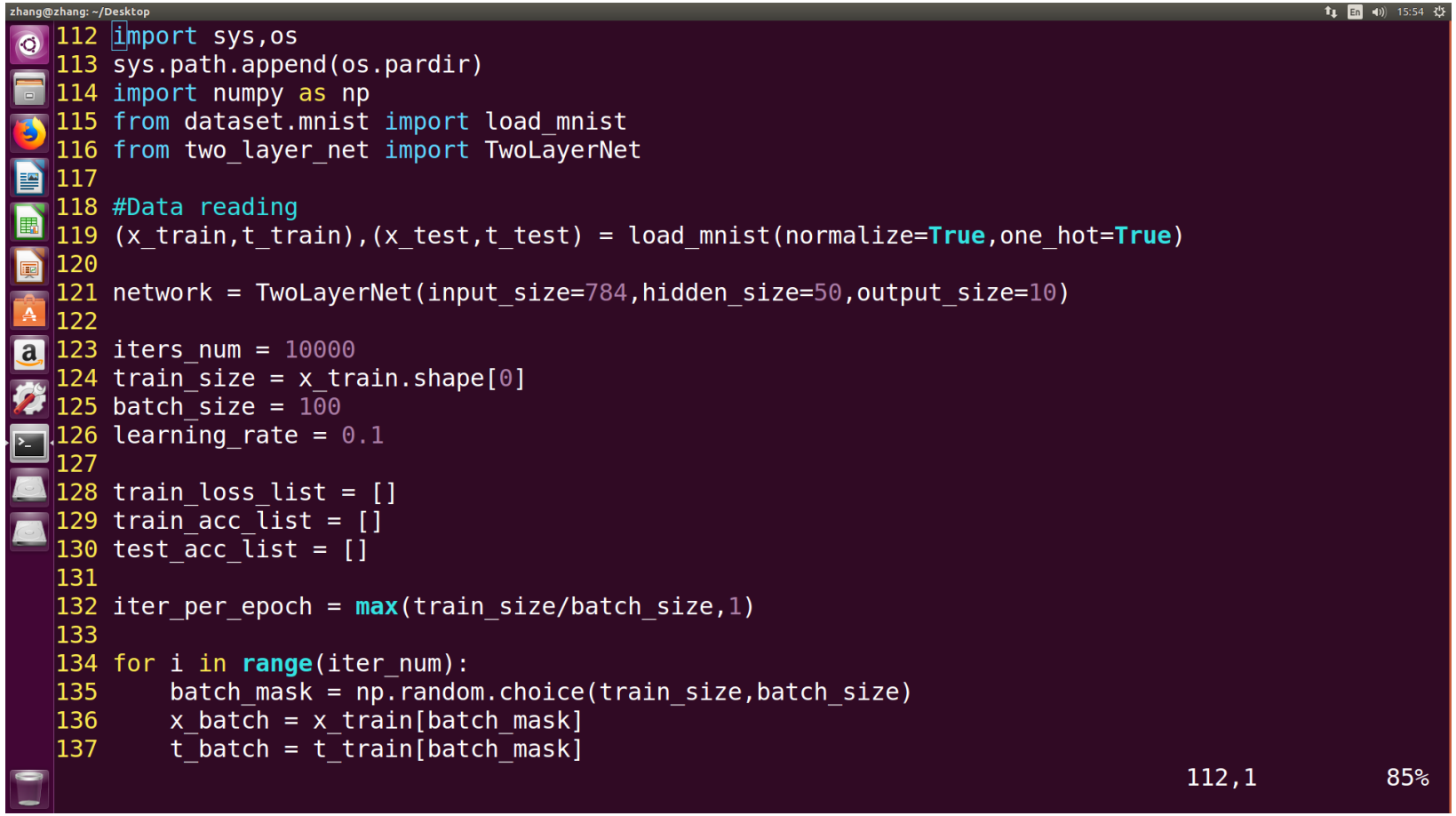


5.7.3 Gradient Calculation and verification

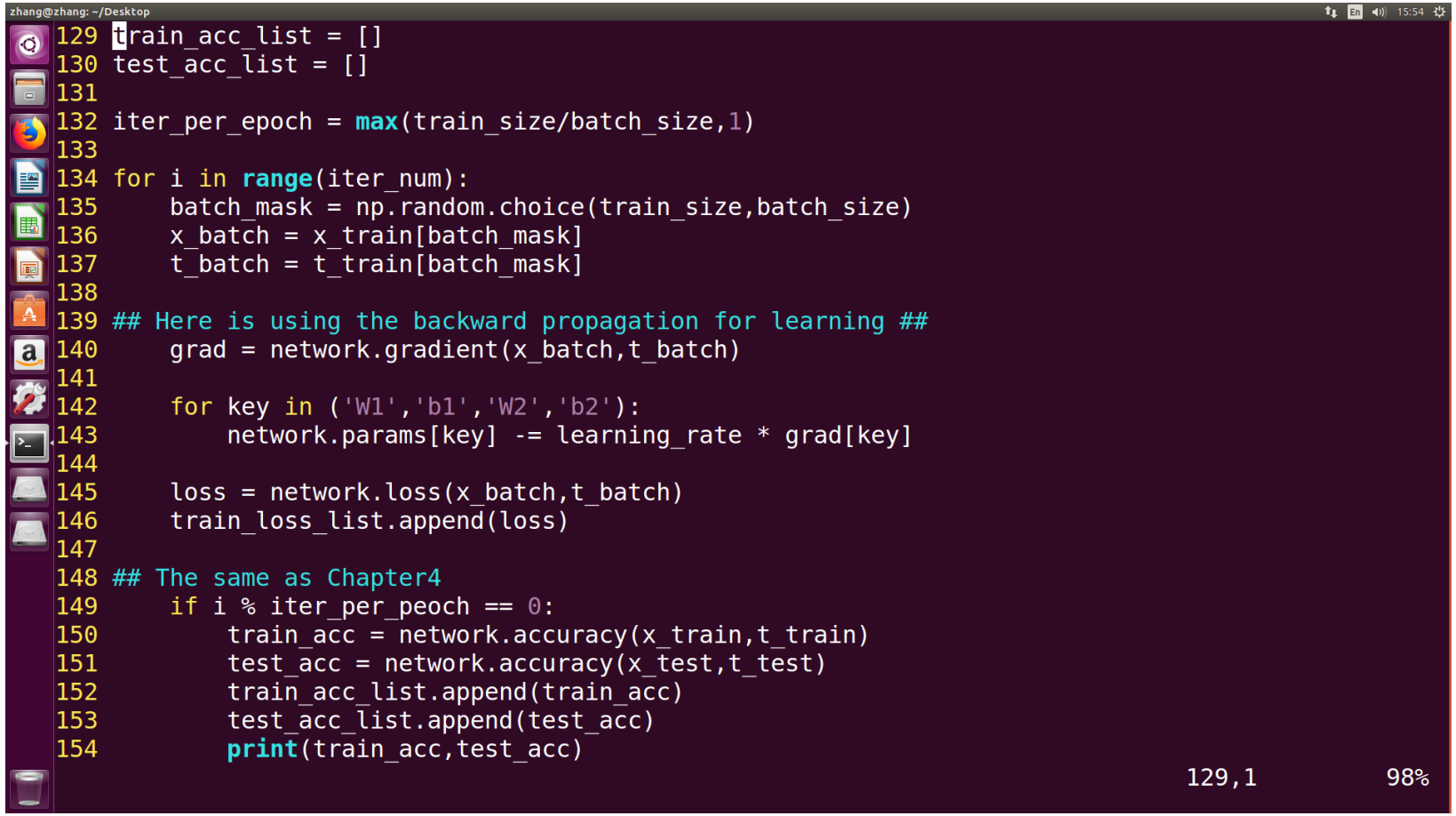



```
b1:9.70418809871e-13
W2:8.41139039497e-13
b2:1.1945999745e-10
W1:2.2232446644e-13
```

5.7.4 Learning by backward propagation network



5.7.4 Learning by backward propagation network



5.8 Summary

1.Know about the calculation process from the computational graph

2.Design the global calculation through the local calculation

3.Know about the calculation result from the forward propagation and get the differential result based on the chain rule.

4.Design the neural network based on the layers and use the backward propagation method to calculate the gradient.

5.Compare the method between the numerical gradient and the backward propagation to confirm the accuracy of the backward propagation

Thanks for your attention