파이썬으로 배우는 딥러닝(Deep Learning)

4회차 수업
딥러닝 학습 로직, 코드로 체험하는 역전파

목 차

퍼셉트론

신경망

신경망학습

오차역전파법

덧셈과 곱셈의 역전파

Affine/Softmax 계층 구현하기

오차역전파법 구현하기

학습관련기술들

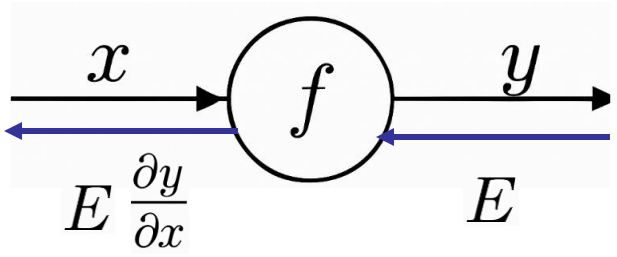
합성곱신경망

전이학습과 ResNet

암석식별머신실습

역전파와 연쇄법칙

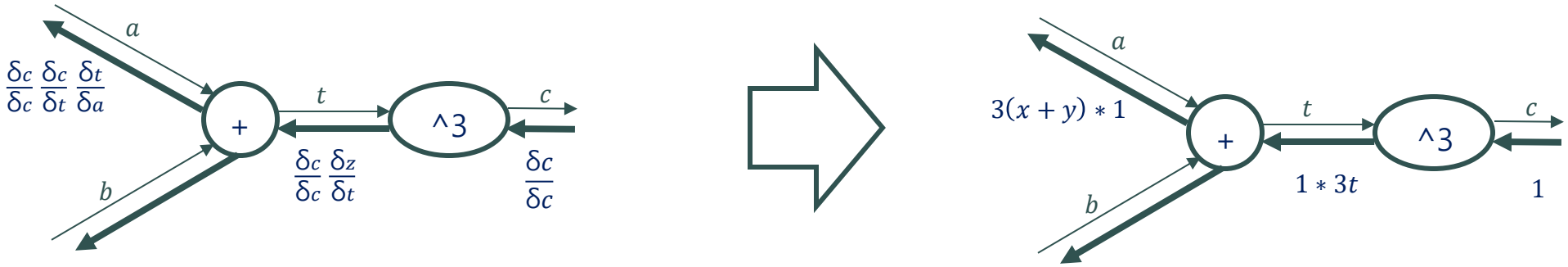
- 계산 그래프의 역전파 : 순방향과는 반대 방향으로 국소적 미분을 곱한다.



- 연쇄 법칙 : 합성함수의 미분 방법

$$\frac{\delta c}{\delta a} = \frac{\delta c}{\delta t}\frac{\delta t}{\delta a} = 3t * 1 = 3(a+b)$$

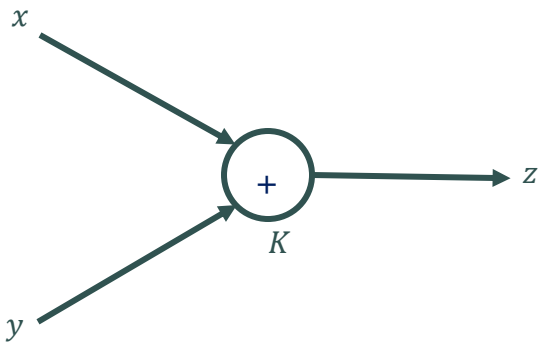
- 합성함수의 미분은 역전파와 동일



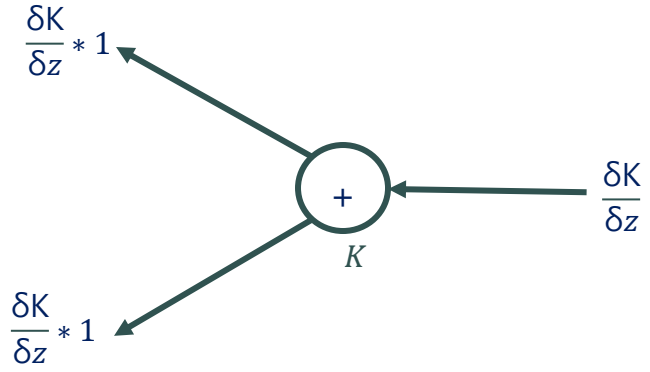
덧셈노드와 곱셈 노드의 역전파

- 덧셈노드의 역전파

$K(z) = K(x + y)$



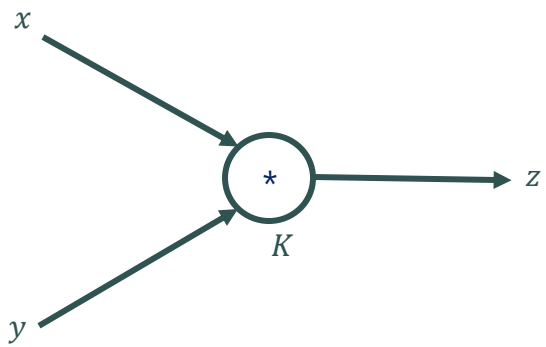
$\frac{\delta}{\delta x}K(x+y) = K'(x+y) * 1$ $\frac{\delta}{\delta y}K(x+y) = K'(x+y) * 1$



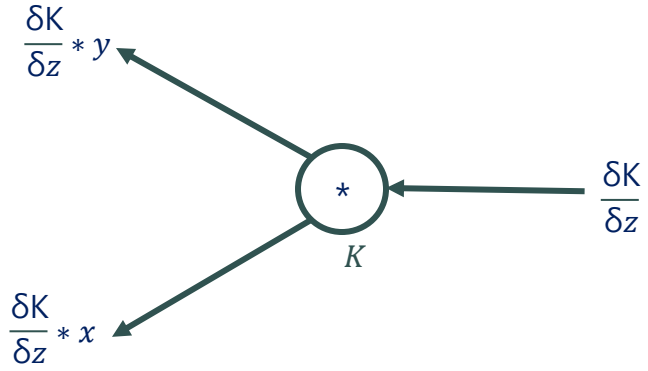
* 덧셈노드의 역전파는 입력값을 그대로 흘려보낸다

- 곱셈노드의 역전파

$K(z) = K(xy)$



$\frac{\delta}{\delta x}K(xy) = K'(xy) * y$ $\frac{\delta}{\delta y}K(xy) = K'(xy) * x$



- 곱셈노드의 역전파는 순전파 때의 입력 신호들을 '서로 바꾼 값'을 곱해서 하류로 흘려보낸다.

목 차

퍼셉트론

신경망

신경망학습

오차역전파법

학습관련기술들

합성곱신경망

전이학습과 ResNet

암석식별머신실습

계산 그래프, 연쇄법칙, 역전파

Affine/Softmax 계층 구현하기

오차역전파법 구현하기

Affine층 계산 그래프

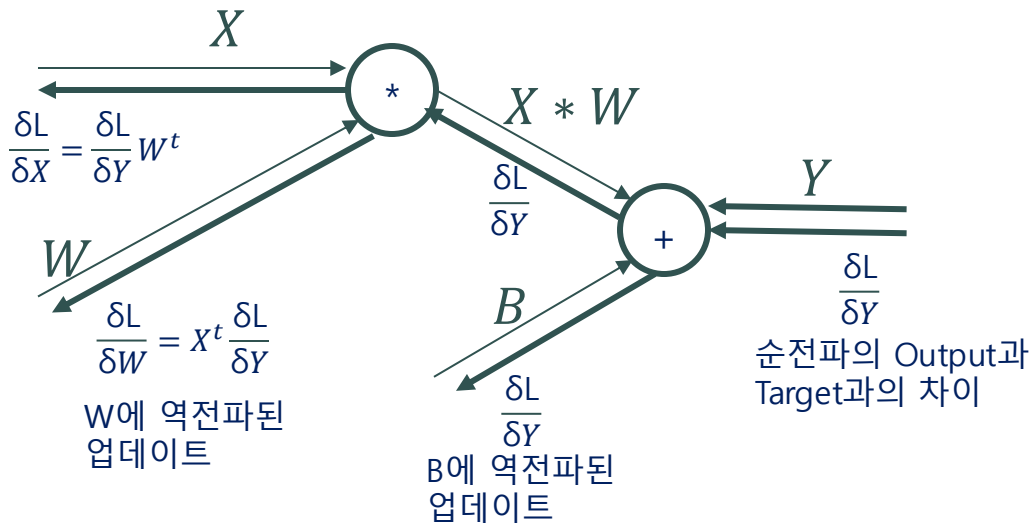
- Affine 층 순전파와 역전파

순전파

$$Y = XW + B$$

$$(y_1 \quad y_2 \quad y_3) = (x_1 \quad x_2)\begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{pmatrix} + (b_1 \quad b_2 \quad b_3)$$

역전파



Sigmoid 층 역전파 그래프

- Sigmoid 층 역전파 간편화 공식 :

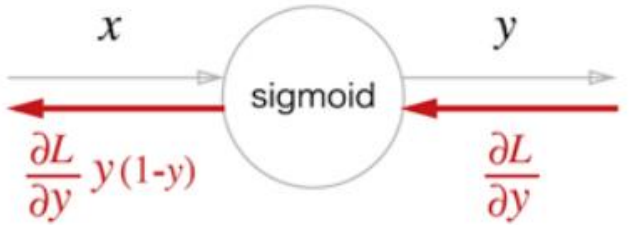
$$y = \frac{1}{1 + e^{-x}} = (1 + e^{-x})^{-1}$$

미분

$$y' = -(1 + e^{-x})^{-2}(-e^{-x})$$

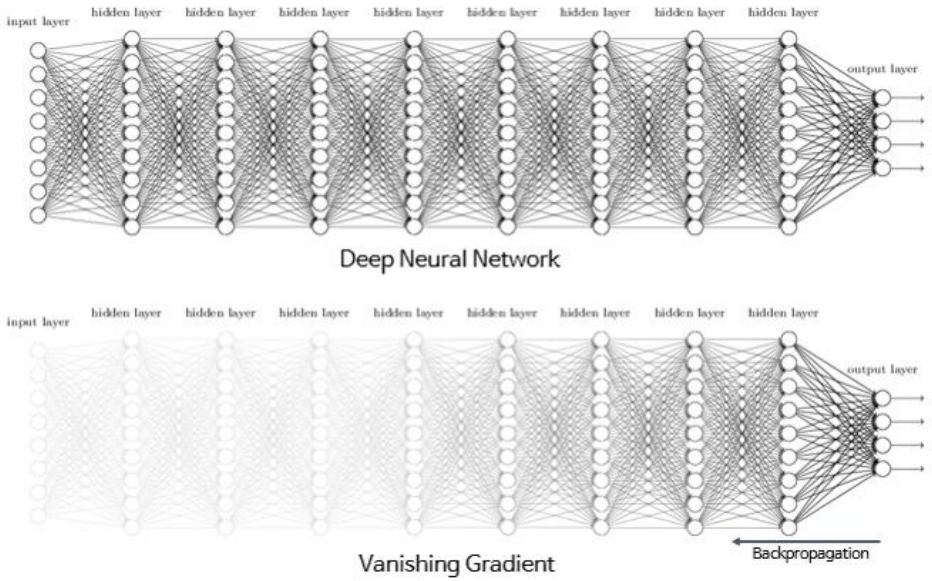
간편화

$$= y(1 - y)$$

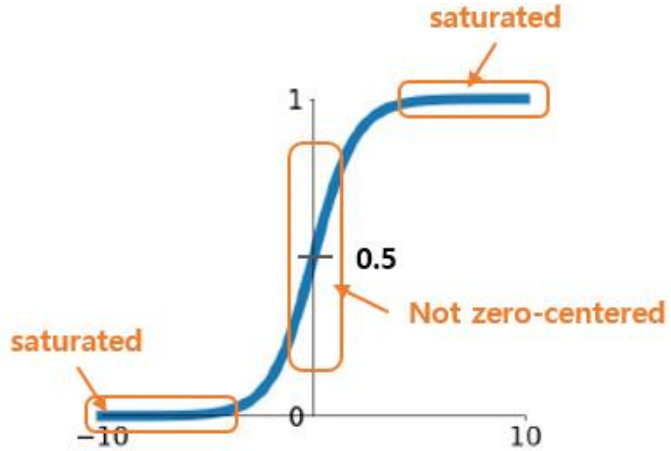


Sigmoid 활성화 함수의 문제점과 대안

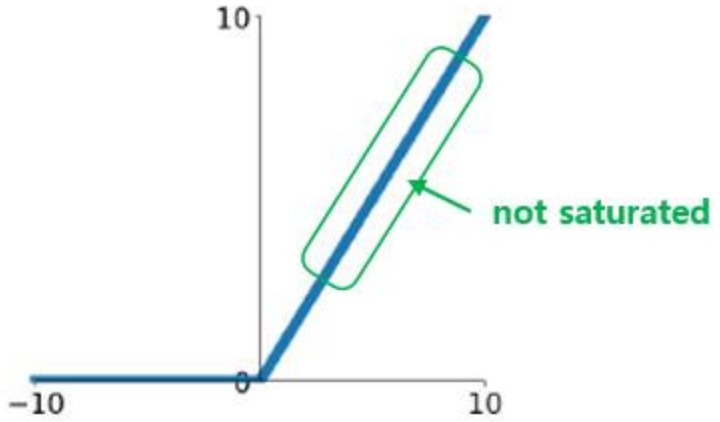
- 역전파(Backpropagation) 과정에서 입력층으로 갈수록 기울기(gradient)가 지수적으로 작아지는 현상



- 원인 : 입력값이 크면 Sigmoid 함수 기울기가 매우 작아져 이전층으로 전달되는 기울기가 점점 작아짐



- 대안 : ReLU함수 사용. 기울기가 1로 유지됨.



ReLU 계층의 순전파와 역전파

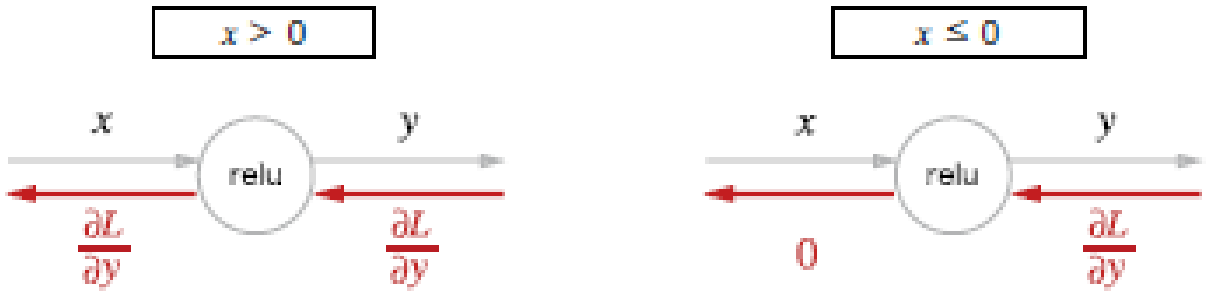
- 활성화 함수로 사용되는 ReLU 수식(순전파)

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

- ReLU 함수의 미분(역전파)

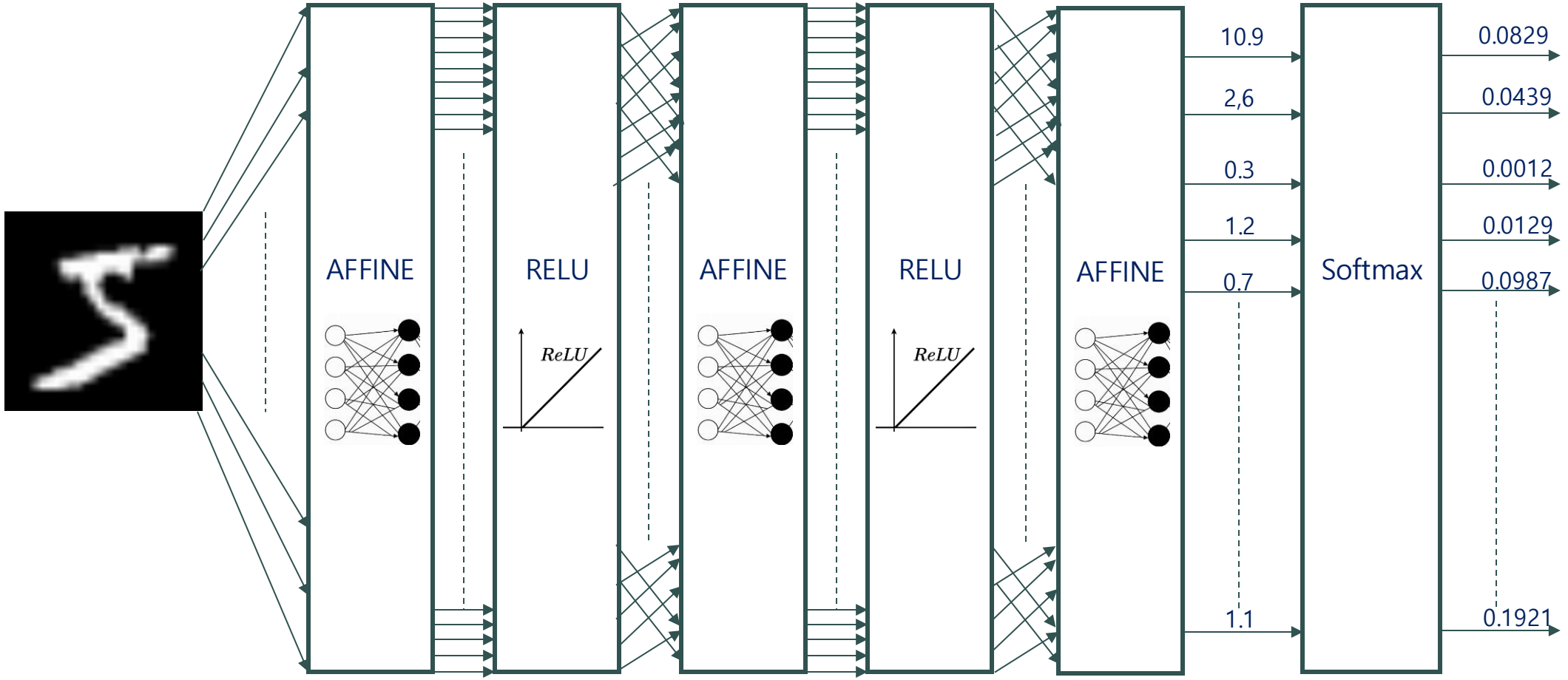
$$\frac{\partial y}{\partial x} = \begin{cases} 1 & (x > 0) \\ 0 & (x \leq 0) \end{cases}$$

- ReLU 함수를 통과하는 역전파



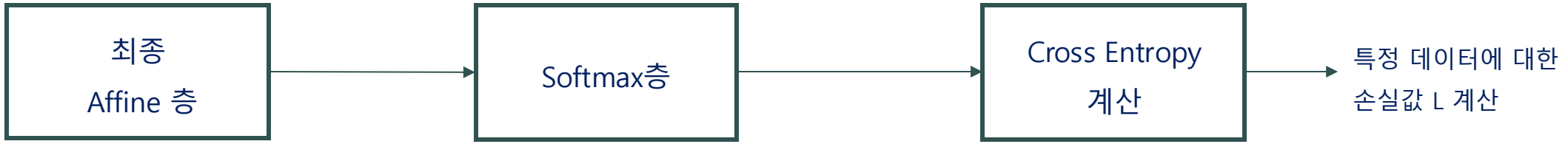
Softmax를 이용한 신경망 추론 과정

- Softmax 함수는 입력 값을 정규화 하여 출력함.



Softmax-with-Loss 층

- Softmax 함수에 손실함수인 Cross Entropy Error 를 이용하여 손실값 구함



- 역전파을 위한 Softmax-with-Loss 통합 미분 함수

$$\frac{\partial L}{\partial a} = (y_1 - t_1, y_2 - t_2, y_3 - t_3) = y - t$$

정답과 실험값의 차이로
미분값을 간단히 구할 수 있음.

목 차

퍼셉트론

신경망

신경망학습

오차역전파법

덧셈과 곱셈의 역전파

Affine/Softmax 계층 구현하기

오차역전파법 구현하기

학습관련기술들

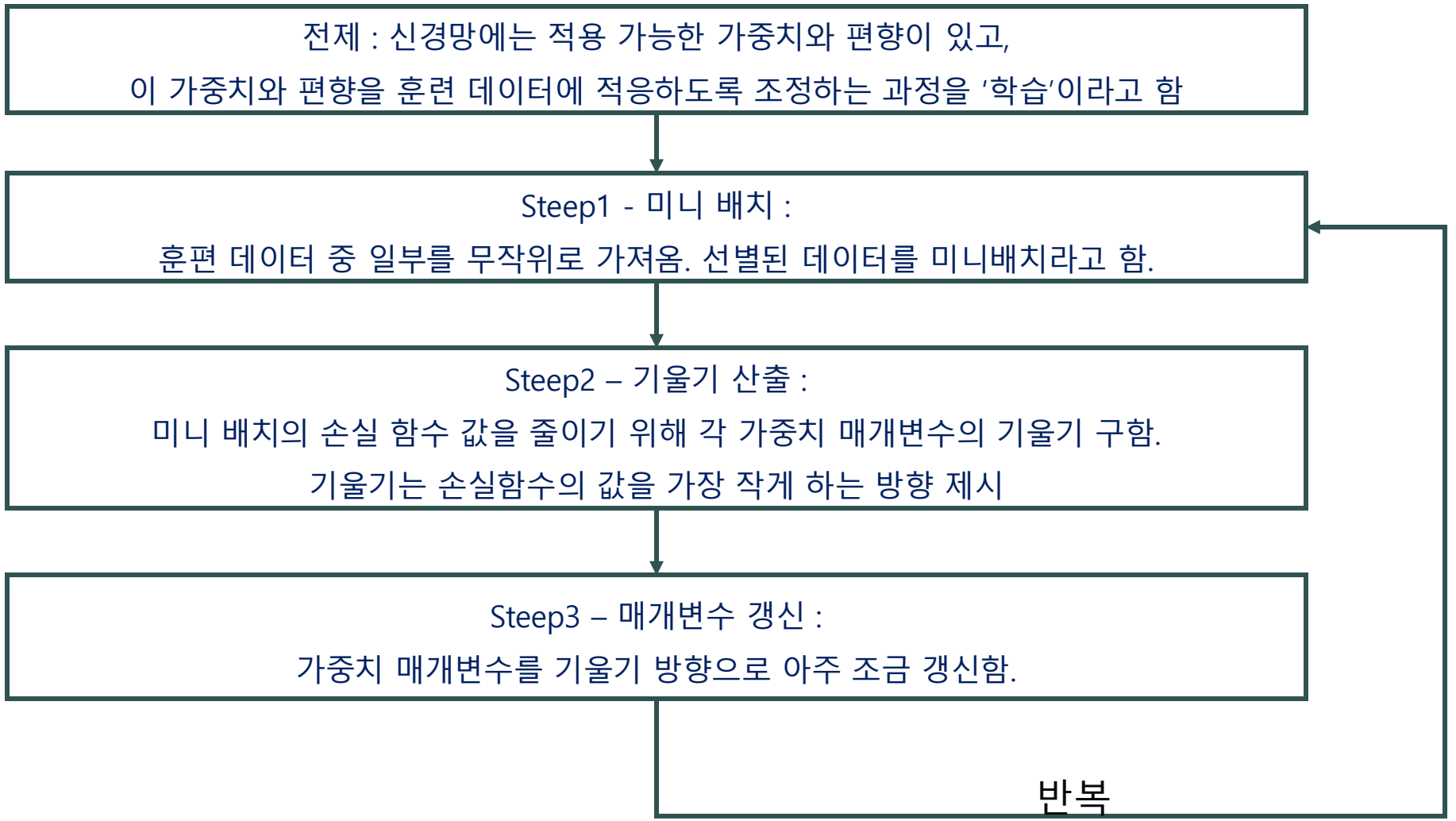
합성곱신경망

전이학습과 ResNet

암석식별머신실습

학습 알고리즘 구현하기

- 학습 알고리즘의 4 단계



TwoLayerNet 클래스 구현하기(1)

- TwoLayerNet 클래스의 인스턴스 변수

인스턴스 변수	설명
params	딕셔너리 변수로, 신경망의 가중치를 보관 - params['W1']은 1-2번층 사이의 가중치, params['b1']은 1-2번층 사이의 편향 - params['W2']는 2-3번층 사이의 가중치, params['b2']는 2-3번층 사이의 편향
layers	OrderedDictionalry 변수로, 신경망의 각 층을 보관
lastLayer	신경망의 마지막 계층 - Softmax와 Loss 계층을 합친 SoftmaxWithLoss 계층
__init__	다음 파라미터로 클래스 초기화를 수행한다. - 입력층 뉴런 수, 은닉층 뉴런 수, 출력층 뉴런 수, 가중치 초기화 시 정규분포의 스케일
predict(self, x)	학습된 Weight를 이용하여 예측을 수행한다.
loss(self, x, t)	손실 값을 구한다. 인수 x는 이미지 데이터, t는 정답 레이블
accuracy(self, x, t)	정확도를 구한다
numerical_gradient	Weight의 기울기를 수치 미분 방식으로 구한다
gradient(self, x, t)	Weight의 기울기를 오차역전파법으로 구한다

<실습 과제>

TwoLayerNet 클래스를

구현해 보자

참고 > ch04/two_layer_net.py 에 작성함.

실습 과제

실습: ch04/two_layer_net.py

```python
# coding: utf-8
import sys, os
current_dir = os.path.dirname(__file__)
parent_dir = os.path.dirname(current_dir)
sys.path.append(parent_dir)

import numpy as np
from common.layers import *
from common.gradient import numerical_gradient
from collections import OrderedDict


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_size)
        self.params['b1'] = np.zeros(hidden_size)
        self.params['W2'] = weight_init_std * np.random.randn(hidden_size, output_size)
        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

```

필요한 Library import

순전파로 신경망 출력을 계산하는 함수

실습 과제

실습: ch04/two_layer_net.py

```
36
37      # x : 입력 데이터, t : 정답 레이블
38      def loss(self, x, t):
39          y = self.predict(x)
40          return self.lastLayer.forward(y, t)
41
42      def accuracy(self, x, t):
43          y = self.predict(x)
44          y = np.argmax(y, axis=1)
45          if t.ndim != 1 : t = np.argmax(t, axis=1)
46
47          accuracy = np.sum(y == t) / float(x.shape[0])
48          return accuracy
49
50      # x : 입력 데이터, t : 정답 레이블
51      def numerical_gradient(self, x, t):
52          loss_W = lambda W: self.loss(x, t)
53
54          grads = {}
55          grads['W1'] = numerical_gradient(loss_W, self.params['W1'])
56          grads['b1'] = numerical_gradient(loss_W, self.params['b1'])
57          grads['W2'] = numerical_gradient(loss_W, self.params['W2'])
58          grads['b2'] = numerical_gradient(loss_W, self.params['b2'])
59
60          return grads
61
```

신경망 출력과 정답을 비교해서 손실값을 계산하는 함수

실제 예측을 했을때 정답을 맞췄는지 정확도를 계산

수치 미분으로 가중치 기울기를 계산하는 함수

실습 과제

실습: ch04/two_layer_net.py

```
61
62      def gradient(self, x, t):
63          # forward
64          self.loss(x, t)
65
66          # backward
67          dout = 1
68          dout = self.lastLayer.backward(dout)
69
70          layers = list(self.layers.values())
71          layers.reverse()
72          for layer in layers:
73              dout = layer.backward(dout)
74
75          # 결과 저장
76          grads = {}
77          grads['W1'], grads['b1'] = self.layers['Affine1'].dW, self.layers['Affine1'].db
78          grads['W2'], grads['b2'] = self.layers['Affine2'].dW, self.layers['Affine2'].db
79
80          return grads
81
```

역전파 방식으로 가중치를 업데이트 하는 함수

손실값 미분함수로 기울기를 구하는 부분

각 Layer의 가중치 기울기를 구하는 부분

역전파 방식으로 구한 가중치를 저장하는 부분

실습 과제

<실습 과제>

MNIST 숫자 데이터를 인식하고 검증하는 코드를 구현해 보자

참고 > ch04/train_neuralnet.ipynb 에 작성함.

실습 과제

실습: ch04/train_neuralnet.ipynb

```
1   # coding: utf-8
2   import os, sys
3   print(os.getcwd())
4   current_dir = os.path.dirname(os.getcwd())
5   print(current_dir)
6   os.chdir(current_dir)
7
8   #sys.path.append(parent_dir)
9
10  #import torch
11  import numpy as np
12  from dataset.mnist import load_mnist
13  from ch04.two_layer_net import TwoLayerNet
14
15  # 데이터 읽기
16  (x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, one_hot_label=True)
17
18  network = TwoLayerNet(input_size=784, hidden_size=50, output_size=10)
19
20  iters_num = 10000
21  train_size = x_train.shape[0]
22  batch_size = 100
23  learning_rate = 0.1
24
25  train_loss_list = []
26  train_acc_list = []
27  test_acc_list = []
28
29  iter_per_epoch = max(train_size / batch_size, 1)
30
```

TwoLayerNet을 import 하여 숫자인식에 활용

MNIST의 숫자 데이터 다운로드 from Internet

입력(784), 은닉층(50), 출력층(10)의 신경망 생성

학습 횟수 10000번 지정

배치(묶음) 크기 100개로 지정

학습률 0.1로 지정

실습 과제

실습: ch04/train_neuralnet.ipynb

```
30
31  for i in range(iters_num):
32      batch_mask = np.random.choice(train_size, batch_size)
33      x_batch = x_train[batch_mask]
34      t_batch = t_train[batch_mask]
35
36      # 기울기 계산
37      #grad = network.numerical_gradient(x_batch, t_batch) # 수치 미분 방식
38      grad = network.gradient(x_batch, t_batch) # 오차역전파법 방식(훨씬 빠르다)
39
40      # 갱신
41      for key in ('W1', 'b1', 'W2', 'b2'):
42          network.params[key] -= learning_rate * grad[key]
43
44      loss = network.loss(x_batch, t_batch)
45      train_loss_list.append(loss)
46
47      if i % iter_per_epoch == 0:
48          train_acc = network.accuracy(x_train, t_train)
49          test_acc = network.accuracy(x_test, t_test)
50          train_acc_list.append(train_acc)
51          test_acc_list.append(test_acc)
52          print(train_acc, test_acc)
```

손실값의 History를 저장

문자인식의 정확도 계산