
밑바닥부터 시작하는 딥러닝 1 **[경사 하강법]**

순천향대학교
컴퓨터시스템연구실

이인규
21.07.05

CHAPTER 4 : 신경망 학습, 경사법

- ◆ 교차 엔트로피 오차[cross entropy error]를 통한 오차 구하기 → $\log x$ 의 그래프를 따름
 - 입력1 : 원-핫 인코딩 정답 레이블
 - 입력2 : 예측한 확률 레이블
 - 출력 : 교차 엔트로피 오차
- ◆ 활성화 함수 : 시그모이드 함수, 소프트맥스 함수
- ◆ 경사 하강법
- ◆ 2층 신경망

Cross Entropy Error

```
def cross_entropy_error(y, t):  
    if y.ndim == 1: # ndim : 차원  
        t = t.reshape(1, t.size) # reshape(1, t.size) : 다차원 배열을 한 차원으로 변경  
        y = y.reshape(1, y.size) # y.size : y 배열의 개수 반환  
  
    # 훈련 데이터가 원-핫 벡터라면 정답 레이블의 인덱스로 반환  
    if t.size == y.size:  
        t = t.argmax(axis=1) # 세로 : 행(0), 가로 : 열(1) 기준으로 최대값을 저장  
  
    batch_size = y.shape[0] # x 개수 반환  
    return -np.sum(np.log(y[np.arange(batch_size), t] + 1e-7)) / batch_size # 로그 반환
```

✓ 0.5s

활성화 함수 : Sigmoid, Softmax

```
✓ def sigmoid(x):  
    return 1 / (1 + np.exp(-x)) # 자연스러운 곡선을 그리는 함수  
✓ 0.2s
```

```
def softmax(x): # 모든 신호에 영향을 준다.  
    if x.ndim == 2: # 차원이 2차원일때  
        x = x.T # array.T : 배열의 행과 열을 바꾸는 메소드  
        x = x - np.max(x, axis=0) # 세로(0) 행 방향의 최대값을 저장  
        y = np.exp(x) / np.sum(np.exp(x), axis=0)  
        return y.T  
  
    x = x - np.max(x) # 오버플로 대책  
    return np.exp(x) / np.sum(np.exp(x))  
✓ 0.4s
```

경사 하강법

경사 하강법

```
def gradient_descent(f, init_x, lr=0.01, step_num=100): # 최적화 하려는 함수, 초깃값, 학습률, 반복횟수
    x = init_x

    for i in range(step_num):
        grad = numerical_gradient(f, x)
        x -= lr * grad

    return x
```

✓ 0.4s

경사 하강법

```
def numerical_gradient(f, x): # 편미분을 벡터로 정리 : 기울기
    h = 1e-4
    grad = np.zeros_like(x)

    for idx in range(x.size):
        tmp_val = x[idx]

        x[idx] = tmp_val + h # (구하고 싶은 값+1) 미분
        fxh1 = f(x)

        x[idx] = tmp_val - h # (구하고 싶은 값-1) 미분
        fxh2 = f(x)

        # 두 미분 값의 평균으로 구하고 싶은 값의 미분 값을 구한다.
        grad[idx] = ( fxh1 - fxh2 ) / ( 2 * h )
        x[idx] = tmp_val

    return grad
```

✓ 0.4s

2층 신경망 구현 (1)

2층 신경망 클래스 구현

class TwoLayerNet:

```
def __init__(self, input_size, hidden_size, output_size, weight_init_std=0.01): # weight_init_std : 학습률
    self.params = {} # 배치처리를 위해 2층의 신경망 구현
    self.params['W1'] = weight_init_std * np.random.randn(input_size, hidden_size) # 학습률에 따른 변수 조정
    self.params['b1'] = np.zeros(hidden_size) # shape는 같지만 모든 요소가 0으로 되어진 배열 생성
    self.params['W2'] = weight_init_std * np.random.randn(hidden_size, output_size) # 학습률에 따른 변수 조정
    self.params['b2'] = np.zeros(output_size) # shape는 같지만 모든 요소가 0으로 되어진 배열 생성

def predict(self, x):
    W1, W2 = self.params['W1'], self.params['W2']
    b1, b2 = self.params['b1'], self.params['b2']

    a1 = np.dot(x, W1) + b1
    z1 = sigmoid(a1) # 1층 활성화함수 : Sigmoid(시그모이드)
    a2 = np.dot(z1, W2) + b2
    y = softmax(a2) # 2층 활성화함수 : SoftMax(소프트맥스)

    return y

def loss(self, x, t): # 오차
    y = self.predict(x)

    return cross_entropy_error(y, t)

def accuracy(self, x, t): # 정확도
    y = self.predict(x)
    y = np.argmax(y, axis=1) # 최대값 반환
    t = np.argmax(t, axis=1) # 최대값 반환

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

2층 신경망 구현 (2)

```
def numerical_gradient(self, x, t): # 경사 하강법
    loss_W = lambda W: self.loss(x, t)

    grads = {}
    grads['W1'] = numerical_gradient(loss_W, self.params['W1'])
    grads['b1'] = numerical_gradient(loss_W, self.params['b1'])
    grads['W2'] = numerical_gradient(loss_W, self.params['W2'])
    grads['b2'] = numerical_gradient(loss_W, self.params['b2'])

    return grads
```

✓ ✓ 0.1s

학습 코드

```

from dataset.mnist import load_mnist # pickle 파일 사용

(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 = []

# 60000 / 100
iter_per_epoch = max(train_size / batch_size, 1)

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]

    grad = network.numerical_gradient(x_batch, t_batch)

    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 | " + str(train_acc) + ", " + str(test_acc))

```

앞으로 목표

- ◆ 현재 배운 내용의 MNIST 파일은 pkl (피클)임
앞으로는 csv 파일을 더욱 많이 사용하게 될 것
pkl 대신 csv 파일을 사용가능하도록 코드를
수정할 것이다.

Question?



Please contact :

이인규
순천향대학교 컴퓨터학부
멀티미디어관 M606

Email : dldlsrb1414@naver.com