HW 5

2021-27764 안지수

문제 1.

1. 학습 시 Forward computation 결과와 정답 label을 비교하는 부분은 어디인가?

```
test_results = [(np.argmax(self.feedforward(x)), y)
                  for (x, y) in test_data]
```

foward computation의 결과는 np.argmax(self.feedforward(x)) 이고, 정답 label은 y 이기 때문에 이부분이 비교하는 부분이다.

2. 가장 마지막 layer(=output layer) output의 gradient를 계산하는 부분은 어디인가?

```
delta = self.cost_derivative(activations[-1], y) * sigmoid_prime(zs[-1])
```

코드의 이 부분에서 activations는 레이어별 h_j값을 나타내고, y는 정답 레이블을 나타낸다. 따라서 self.cost_derivative(activations[-1], y) 부분이 output 마지막 레이어의 아웃풋 값인 h로 error를 미분하는 gradient를 계산하는 값이다.

3. 상기한 gradient로부터 마지막에서 두 번째 layer(=마지막 hidden layer) output의 gradient를 계산하는 부분은 어디인가? 코드 위치, 해당 코드에서 각 변수의 의미, 연산의 의미를 설명할 것.

```
delta = np.dot(self.weights[-l + 1].transpose(), delta) * sp
```

이 부분에서 `l` 값이 2일때가 마지막에서 두번째 레이어의 output gradient를 계산하는 식이다. 이 부분인 이유는 실제로 미분을 진행해보면 chain-rule에 따라, weight와 delta 그

리고 sigmoid prime를 곱했을 때의 값이 gradient 값이기 때문에 이 연산을 수행한다. 이 때 차원을 맞추기 위해 weight값에 transpose를 수행한다.

4. 마지막 layer의 weight의 gradient를 계산하는 부분은 어디인가?

```
nabla_w[-1] = np.dot(delta, activations[-2].transpose())
```

activations[-2]은 마지막에서 두번째 레이어의 forward computation의 activation을 수행하기 전 결과이다. 그리고 마지막 layer의 weight의 gradient를 구하기위해 delta와 마지막에서 두번째 레이어의 h_j값을 곱해주어야 하기 때문에 이부분이 마지막 layer의 weight의 gradient를 계산하는 부분이다.

5. 각 layer의 weight를 update하는 부분은 어디인가?

```
nabla_w[-1] = np.dot(delta, activations[-2].transpose())
nabla_w[-l] = np.dot(delta, activations[-l - 1].transpose())
```

이 두 부분이 각 레이어의 weight를 업데이트 해주는 부분이다 위의 4번문제에서 설명한 방식으로, 각 w_j의 gradient를 구하기위해 delta와 h_j값을 곱해주어야 하기 때문에 이 위 두 부분이 각 레이어들의 weight를 업데이트 하는 부분이다.

문제 2.

```
# backward pass
delta = self.cost_derivative(activations[-1], y) * sigmoid_prime(zs[-1])
nabla_b[-1] = delta
nabla_w[-1] = np.dot(delta, activations[-2].transpose())

# if fine-tuning
if tuning == True:
  return (nabla_b, nabla_w)

for l in range(2, self.num_layers):
    z = zs[-l]
    sp = sigmoid_prime(z)
    delta = np.dot(self.weights[-l + 1].transpose(), delta) * sp
```

```
    nabla_b[-l] = delta
    nabla_w[-l] = np.dot(delta, activations[-l - 1].transpose())
```

weight를 업데이트하는 부분인 backward 부분을 살펴보면 먼저 마지막 레이어의 weight 값을 업데이트 하고 이후에 나머지 부분의 weight를 업데이트 한다.

이때 fine-tuning을 지원하기 위해 fine-tuning에 ture를 주었을 때, 마지막 레이어를 제외한 나머지 레이어의 weight값의 업데이트를 스킵하도록 디자인 했다.

문제 3.

```
# attribute에 추가
self.w_v = [np.zeros(w.shape) for w in self.weights]
self.b_v = [np.zeros(b.shape) for b in self.biases]

# update_minibatch에 추가
self.w_v = np.multiply(momentum, self.w_v) + nabla_w
self.weights = [w - (lr / len(mini_batch)) * nw for w, nw in zip(self.weights, self.w_v)]
self.b_v = np.multiply(momentum, self.b_v) + nabla_b
self.biases = [b - (lr / len(mini_batch)) * nb for b, nb in zip(self.biases, self.b_v)]
```

momentum SGD 기능을 구현하기 위해 속도 값을 저장하고 이를 minibatch를 업데이트 하는 부분에 추가하여 구현했다.