연구지도 및 연구윤리 10주차 보고서

(Chapter4~5. 신경망, 오차역전파)

경북대학교 전자공학부
2016113566 김남영

1) 신경망 학습 절차(단계)

1단계 - 미니배치

모든 훈련 데이터를 대상으로 손실 함수 값을 구해서 더하고, 정규화를 통해 평균 손실 함수를 구해야한다. 학습의 목표는 평균 손실 함수를 줄이는 방향으로 매개변수(가중치와 편향)를 반복해서 갱신하는 것이다. 하지만 훈련 데이터가 빅데이터 수준이 되면, 일일이 손실 함수를 계산 하는 것은 비현실적이므로, 훈련 데이터 중 일부만을 무작위로 가져와서 학습을 수행하고, 이를 전체의 근사치로 활용한다. 이러한 학습 방법을 미니배치라고 한다.

2단계 - 기울기 산출

기울기(gradient)는 모든 매개변수의 편미분을 벡터로 정리한 것을 의미한다. 여기서 매개변수는 가중치와 편향을 의미하고, 손실 함수가 작아지는 방향으로 변수를 갱신해야한다. 이 때, 기울기가 가리키는 벡터의 방향은 함수의 값을 낮추는 방향이므로, 손실 함수 f의 값이 작아질 수 있는 매개변수의 기울기를 구할 수 있게 된다.

3단계 - 매개변수 갱신

매개변수의 기울기 값을 구했다면, 갱신하는 정도를 의미하는 학습률을 기울기 값에 곱한 뒤 원래 매개변수 값에서 빼주면 새로운 값으로 갱신이 된다. 이 새로운 갱신 값은 손실 함수를 줄이는 방향으로 갱신된다(기울기의 의미가 함수의 값을 낮추는 방향의 벡터를 의미하므로). 의미상 기울기를 줄이는 계산을 반복하므로 이를 경사하강법이라 말한다.

4단계 - 반복

위 과정을 반복하여, 손실 함수를 줄여나가면 학습이 완료된다. 손실을 줄인다는 말은 곧 정확도를 높인다는 의미이므로 반복을 할수록 정확도 높은 학습 모델이 구현이 된다.

5단계 - 평가

학습의 궁극적 목표는 범용적 능력을 익히는 것이므로, 훈련 데이터에 포함되지 않는 시험 데이터를 사용하여 평가해보아야 한다. 따라서, 학습 도중 정기적으로 훈련 데이터와 시험 데이터를 대상으로 정확도를 기록하여, 오버피팅을 일으키지 않고 잘 학습 되고 있는지를 확인해준다. 미니배치를 반복하여, 훈련 데이터를 모두 소진하는 횟수를 1에폭이라고 정의하는데, 1에폭마다 정확도를 계산해서 훈련 데이터와 시험데이터의 정확도에 차이가 없는지를 확인할 수 있다.

2) 예제 코드(4.5.2, 4.5.3) 한 줄씩 설명 및 분석

2.1) 4.5.2 분석

```python
import numpy as np # numpy를 np라는 이름으로 사용
from dataset.mnist import load_mnist # mnist dataset 불러오기
from two_layer_net import TwoLayerNet # 만들어뒀던 2층 신경망 사용

# mnist dataset 읽기, 데이터를 0.0 ~ 1.0 범위의 데이터로 정규화를 하고,
# 정답 원소만 1값을 가지는 원-핫 인코딩 사용
(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, one_hot_label=True)

train_loss_list = [] # 손실 함수 값의 추이를 확인하기 위해 손실 값을 담는 리스트를 생성

# 하이퍼 파라미터
iters_num = 10000  # 갱신을 반복하는 횟수
train_size = x_train.shape[0]
batch_size = 100   # 미니배치 크기
learning_rate = 0.1 # 학습률 설정

# 이전에 구현한 2층 신경망 생성
network = TwoLayerNet(input_size=784, hidden_size=50, output_size=10)

# 매개변수 갱신 과정
for i in range(iters_num): # 10000번 갱신 과정 반복
    # 미니배치 획득
    batch_mask = np.random.choice(train_size, batch_size)
    # 6만개의 training dataset 중 100개의 dataset을 무작위로 선정
    x_batch = x_train[batch_mask] # 훈련 데이터 미니 배치 획득
    t_batch = t_train[batch_mask] # 정답 레이블 미니 배치 획득

    # 기울기 계산
    grad = network.numerical_gradient(x_batch, t_batch)
    # 훈련 데이터와 정답 레이블을 비교하여 손실 함수를 계산하고,
    # 손실 함수에 대한 매개 변수 기울기를 각각 구함.

    # 매개변수 갱신
    for key in ('W1', 'b1', 'W2', 'b2'):
        network.params[key] -= learning_rate * grad[key]
    # 학습률(갱신하는 정도)을 기울기에 곱하여 원래 매개변수 값에서 뺌
    # 변화 방향은 손실 함수를 낮추는 방향이므로 갱신할수록 손실 함수가 작아진다.

    # 학습 경과 기록
    loss = network.loss(x_batch, t_batch) # 손실 함수 값을 구한다.
    train_loss_list.append(loss)
    # 각 반복마다 손실 함수 값을 리스트로 이어 붙여 추후에 도식적으로 확인하기 위함
```

2.2) 4.5.3 분석

```python
import numpy as np # numpy를 np라는 이름으로 사용
from dataset.mnist import load_mnist # mnist dataset 불러오기
from two_layer_net import TwoLayerNet # 만들어뒀던 2층 신경망 사용

# mnist dataset 읽기, 데이터를 0.0 ~ 1.0 범위의 데이터로 정규화를 하고,
# 정답 원소만 1값을 가지는 원-핫 인코딩 사용
(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, one_hot_label=True)

# 이전에 구현한 2층 신경망 생성
network = TwoLayerNet(input_size=784, hidden_size=50, output_size=10)

# 하이퍼파라미터
iters_num = 10000  # 갱신을 반복하는 횟수
train_size = x_train.shape[0]
batch_size = 100   # 미니배치 크기
learning_rate = 0.1 # 학습률 설정

train_loss_list = []  # 손실 함수 값의 추이를 확인하기 위해 손실 값을 담는 리스트를 생성
train_acc_list = [] # 에폭을 반복할 때마다 training dataset의 accuracy를 계산하여 담는 리스트를 생성
test_acc_list = [] # 에폭을 반복할 때마다 test dataset의 accuracy를 계산하여 담는 리스트를 생성

# 1에폭당 학습수(6만개의 training set 중 100개의 미니배치 => 1에폭은 미니배치의 600번 반복)
iter_per_epoch = max(train_size / batch_size, 1)

# 매개변수 갱신 과정
for i in range(iters_num):
    # 미니배치 획득
    batch_mask = np.random.choice(train_size, batch_size)
    # 6만개의 training dataset 중 100개의 dataset을 무작위로 선정(임의의 정수)
    x_batch = x_train[batch_mask] # 훈련 데이터 미니 배치 획득
    t_batch = t_train[batch_mask] # 정답 레이블 미니 배치 획득

    # 기울기 계산
    grad = network.numerical_gradient(x_batch, t_batch)
    # 훈련 데이터와 정답 레이블을 비교하여 손실 함수를 계산하고,
    # 손실 함수에 대한 매개 변수 기울기를 각각 구함.

    # 매개변수 갱신
    for key in ('W1', 'b1', 'W2', 'b2'):
        network.params[key] -= learning_rate * grad[key]
    # 학습률(갱신하는 정도)을 기울기에 곱하여 원래 매개변수 값에서 뺌
    # 변화 방향은 손실 함수를 낮추는 방향이므로 갱신할수록 손실 함수가 작아진다.

    # 학습 경과 기록
    loss = network.loss(x_batch, t_batch) # 손실 함수 값을 구한다.
    train_loss_list.append(loss)
    # 각 반복마다 손실 함수 값을 리스트로 이어 붙여 추후에 도식적으로 확인하기 위함

    # 1에폭당 정확도 계산(600번째 갱신마다 정확도 계산)
    if i % iter_per_epoch == 0:
        train_acc = network.accuracy(x_train, t_train) # training set의 정확도를 구함
        test_acc = network.accuracy(x_test, t_test) # test set의 정확도를 구함
        train_acc_list.append(train_acc) # 도식화를 위해 리스트에 정확도를 넣음
        test_acc_list.append(test_acc) # 도식화를 위해 리스트에 정확도를 넣음
        print("train acc, test acc | " + str(train_acc) + ", " + str(test_acc))
```

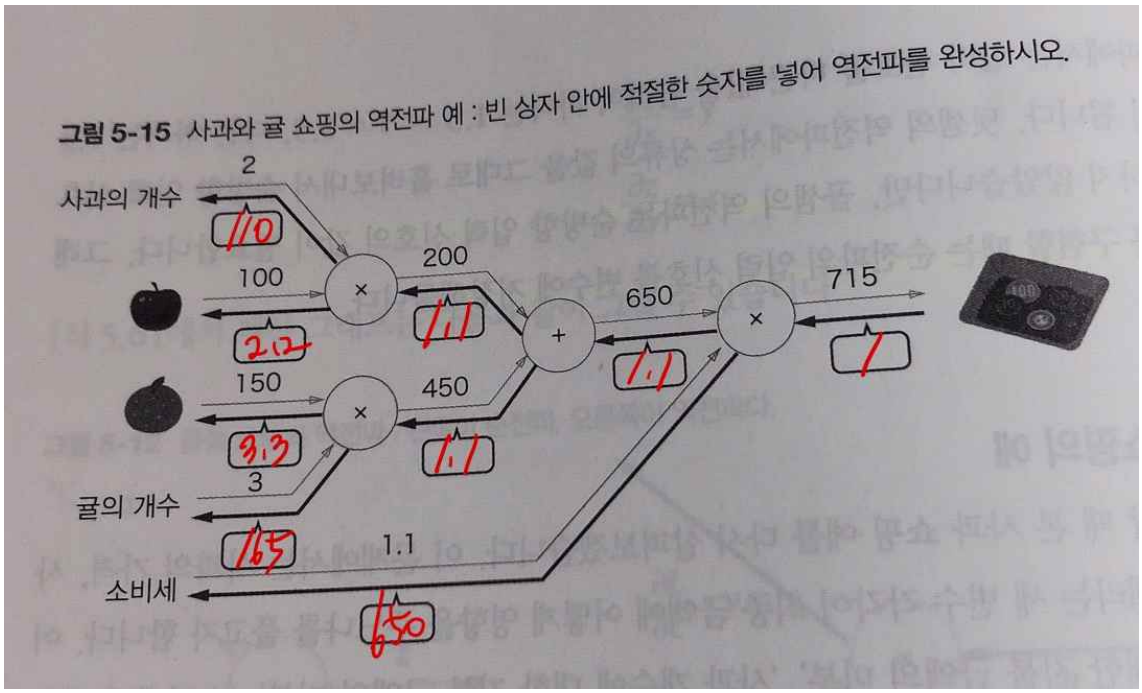
3) 오차역전파에 대한 설명 요약

- 오차역전파

매개변수의 기울기를 수치 미분으로 구할 수 있지만 계산 시간이 오래 걸린다는 단점이 있다. 이때 이를 효율적으로 계산하는 방법이 오차역전파법이다. 계산 그래프에서 왼쪽에서 오른쪽으로 순방향으로 계산을 진행하는 것을 순전파라고 하고, 이를 역방향으로 진행하는 것을 역전파라고 한다. 역전파는 미분 값을 전달하므로, 입력이 조금 변했을 때 출력이 얼마나 변화

하는지를 쉽게 계산할 수 있다. 따라서 입력 값이 조금 변했을 때 정답 레이블과의 오차를 역전파를 통해 쉽게 계산할 수 있다. 덧셈 노드의 경우 입력 값(역방향의 입력 값)을 그대로 출력하고, 곱셈 노드의 경우 역방향의 입력 값에 순방향의 입력 값을 서로 뒤바꾼 값을 곱하여 출력한다.

- 그림 5-14 사과 귤 쇼핑 값 직접 계산해보기



덧셈 노드와 곱셈 노드의 성질을 이용하면 쉽게 역전파 값을 구할 수 있다. 귤, 사과의 가격과 소비세가 마지막 곱셈 노드로 묶여 있으므로, 귤, 사과의 가격과 소비세가 같은 양만큼 오르면 사과 가격은 2.2의 크기, 귤의 가격은 3.3의 크기, 소비세는 650의 크기만큼 영향을 받는다고 해석할 수 있다.

고찰

1. 여태껏 보고서를 써오면서 오차역전파법이라는 말을 많이 들었었는데 이제 그 기초를 알게 되어 뿌듯하다.
2. 사과 예제를 통해 오차역전파법을 이해했지만, 실제로 손실 함수의 값을 줄일 때 어떤 방식으로 사용되는지는 아직 모르겠다.
3. 다른 여러 전공 과목에서 gradient를 많이 봤음에도 이해가 쉽지 않았는데, 쉽고 도식적으로 이해할 수 있어서 다행이다.
4. 자동으로 매개변수를 설정 한다는게 이전에는 마법 같은 일처럼 느껴졌는데 생각보다 단순한 원리를 통해 이루어진다는 것을 알게되었다.
5. 쉽게 이해하는 것도 중요하지만 수식적으로도 이해할 수 있도록 노력해야겠다.